ALESSANDRO MAZZETTI

Stato dell'Arte

Una nuova sfida tecnologica

Reti Neurali Artificiali

- Cervello biologico e simulazioni
- Neuroni formali e reti artificiali
- Memoria associativa distributiva
- Reti Back Propagation
- Applicazioni e strumenti

Introduzione ai principali modelli e simulazione su Personal Computer

APOGEO

RAI TELEVIDEO

ALESSANDRO MAZZETTI

Una nuova sfida tecnologica

Reti Neurali Artificiali

Introduzione ai principali modelli e simulazione su personal computer

APOGEO

Reti Neurali Artificiali
Introduzione ai principali modelli
e simulazione su personal computer

Autore
Alessandro Mazzetti

Copyright © 1991

RAI - Telesoftware
Piazza Montegrappa 4
00195 Roma
Telefono (06) 3220686
Fax (06) 3225427

APOGEO - Editrice di Informatica
Via Voghera 11/A
20144 Milano
Telefono (02) 89404722, 89408423
Fax (02) 89404595

ISBN 88-7303-002-5

Grafica, copertina
e impaginazione elettronica di Anna Nicoli e Mauro Risani

Tutti i diritti sono riservati a norma di legge ed a norma delle convenzioni internazionali. Nessuna parte di questo libro può essere riprodotta con sistemi elettronici, meccanici, o altri senza l'autorizzazione scritta dell'Editore.

Prefazione

Questo libro riproduce la parte testuale della versione elettronica di un corso commissionato all'autore dalla Divisione Televideo della RAI e trasmesso nell'ambito del proprio servizio Telesoftware.

Nato nel 1984, Televideo ha ampliato dalla fine del 1990 la gamma della sua offerta e ha avviato un servizio di trasmissione dati via etere, cioè di software per personal computer, dedicato all'utenza domestica, che affianca le trasmissioni dati, sempre via etere, per utenze commerciali e per gruppi chiusi di utenza e la trasmissione di software didattico per il mondo della scuola. Il servizio dedicato all'utenza domestica è stato denominato "Personal Software" e mette a disposizione, gratuitamente, di tutti i possessori di un personal computer con sistema operativo MS-DOS materiali informatici di varia natura.

Le pagine di Televideo sono suddivise in otto indici e sono contenute in otto magazzini, sette dei quali vengono utilizzati per la trasmissione cosiddetta "in chiaro" di informazioni e notizie.

L'ottavo magazzino, quello che comprende le pagine da 0 a 99 (per alcune marche di televisori da 800 a 899), è stato riservato alle trasmissioni del Telesoftware, alla trasmissione via etere, cioè, di file di qualsivoglia tipologia per calcolatori: programmi, dati, testi, grafica.

In pratica, ogni utente, installando nel proprio calcolatore una apposita scheda Televideo/Telesoftware da collegare alla presa di antenna TV, può "ricevere" via etere senza alcun onere aggiuntivo il software trasmesso, memorizzarlo nel

computer e quindi utilizzarlo secondo i tradizionali criteri informatici.

L'offerta di programmi di "Personal Software" è molto ampia ed è soggetta a continui incrementi. In particolare, nell'arco dei primi mesi di programmazione sono state trasmesse rubriche dedicate a libri elettronici, a programmi didattici, ad applicazioni cosiddette di "produttività personale", a notiziari informativi, a software di pubblico dominio e autoprodotto. Molte di queste rubriche, e di quelle in fase di allestimento, sono realizzate in collaborazione con altre aziende che operano o che si avviano a operare nel mondo dell'editoria elettronica e della diffusione dati.

La pagina 750 del Televideo offre un indice delle diverse aree di programmazione e fornisce informazioni sui programmi trasmessi, sulle loro caratteristiche essenziali e sulle modalità per riceverli.

Ringraziamenti

L'Autore desidera ringraziare tutte le persone che hanno contribuito allo sviluppo di questo libro, in particolare Pasquale Santoli, sostenitore della versione "via etere" del presente libro, trasmessa dalla rubrica Telesoftware di RAI-Televideo. Un particolare ringraziamento a Bruno Apolloni, Elisabetta Binaghi, Gabriele Ronchini e Gilberto Vanzetto per avermi seguito durante la realizzazione del libro con i loro pregiati consigli.

L'Autore

Alessandro Mazzetti, dopo aver conseguito la laurea in fisica-cibernetica a Milano nel 1980, si è occupato di linguaggi e metodologie presso l'Etnoteam SpA e la Direzione Centrale delle Ricerche dell'Italtel SpA. Nel 1985 si è avvicinato all'Intelligenza Artificiale operando come "Knowledge Engineer" presso l'Expert Systems Application Centre della Systems Designers SpA. È autore di numerose pubblicazioni fra le quali tre libri sui Sistemi Esperti e sul Prolog; la sua passione per lo studio delle più avanzate tecnologie si esprime attraverso la continua collaborazione con l'Università di Milano nel campo dei Sistemi che Apprendono e delle Reti Neurali. Attualmente è responsabile del Centro Tecnologie Innovative della CAP-Gemini-Sysdata SpA, dove ricopre il ruolo di Product Manager per i Sistemi Esperti e i Prodotti Neurali.

A Jacopo e Ruggero

Sommario

1 **Il cervello biologico e le sue simulazioni** 1

2 **Neuroni formali e reti artificiali** 9

3 **Computabilità, apprendimento e caratteristiche** 15

4 **Memoria associativa distribuita** 23

5 **Simulazione software di una memoria associativa** .. 37

6 **Reti "back propagation"** 55

- *Appendice 65*

7 **Applicazione di una rete back propagation** 69

8 **Altri modelli di rete neurale** 89

- *Modelli stocastici 89*
- *Modelli auto-organizzanti 93*
- *Modelli genetici 97*

9 **Applicazioni e strumenti** 99

- *Strumenti di sviluppo per Reti Neurali 102*

Bibliografia 107

1 Il cervello biologico e le sue simulazioni

Tre sono le discipline rivolte allo studio dei fenomeni intellettuali:

- la *neurologia*, orientata a capire *come è fatto* il cervello, vale a dire quali sono i componenti biologici interni, qual è la struttura fisica e come curare dal punto di vista chirurgico le possibili malattie.
- la *psicologia*, orientata a capire *come si comporta* il cervello, quindi gli effetti del suo funzionamento, indipendentemente dai processi fisico-chimici che avvengono al suo interno.
- *l'intelligenza artificiale*, orientata a *emulare i comportamenti intelligenti* attraverso dispositivi meccanico-elettronici, generalmente di natura molto differente da quelli biologici.

Fino a qualche anno fa queste tre discipline hanno proseguito la loro strada senza influenzarsi a vicenda, giungendo a risultati interessanti e utili ma spesso insoddisfacenti [12]. In parole povere, a tutt'oggi non sappiamo come funziona il cervello umano; non siamo in grado di spiegare adeguatamente i processi di apprendimento; non siamo capaci di costruire macchine che possano essere scambiate per esseri viventi.

Sembra che per superare questi scogli sia necessaria una cooperazione e una sinergia fra diversi rami di studio. Nasce quindi una nuova disciplina che unisce neurologia, psicologia, intelligenza artificiale, biologia, ingegneria, fisica e matematica con lo

scopo di progredire nello studio dei fenomeni intellettuali. Non si tratta solo di voler costruire macchine più intelligenti del computer, ma anche di far progredire la psicologia e la neurologia mediante simulazioni che fungano da esperimenti di laboratorio per le teorie da convalidare.

Svariati nomi sono stati attribuiti a questa nuova disciplina: *Neuroinformatica, Paradigma Sub-simbolico, Connessionismo, Parallel Distributed Processing* (queste ultime due sono discipline più generali in cui le reti neurali sono un possibile approccio). Negli ambienti informatici, questa disciplina ha inizialmente esordito nel tentativo di riprodurre le strutture nervose dei tessuti cerebrali su strumenti di calcolo, ma sta man mano progredendo con l'identificazione di modelli matematici che hanno sempre meno a che fare con la biologia, la psicologia o la medicina. A tutt'oggi, col termine *Reti Neurali* (o *Neuronali* o *Neuroniche* [13]) si identifica una tecnologia di elaborazione dell'informazione complementare all'informatica classica.

Il cervello umano ha circa 10 bilioni di cellule nervose (neuroni), connesse fra di loro in vario modo (figura 1). Il numero di interconnessioni è stimato dell'ordine di un milione di miliardi.

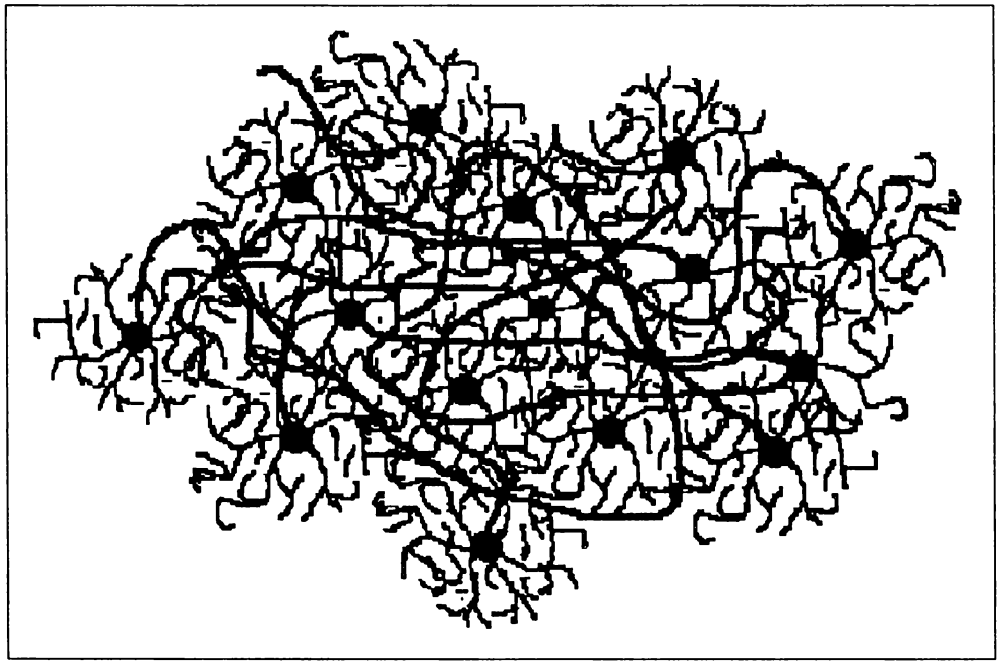

Figura 1. *Rete di neuroni biologici*

Quando un neurone viene attivato, manda un impulso elettrochimico ai neuroni cui è connesso: l'operazione si ripete per ogni neurone e nell'intervallo di pochi centesimi di secondo intere regioni del cervello risultano interessate.

Il neurone è il componente elementare del sistema nervoso; vi sono molti tipi di neuroni, ma nell'uomo non sono presenti tipi diversi da quelli di altre specie animali [1]. Il neurone, come mostra la figura 2, è composto da:

- corpo cellulare;
- dendriti;
- assone.

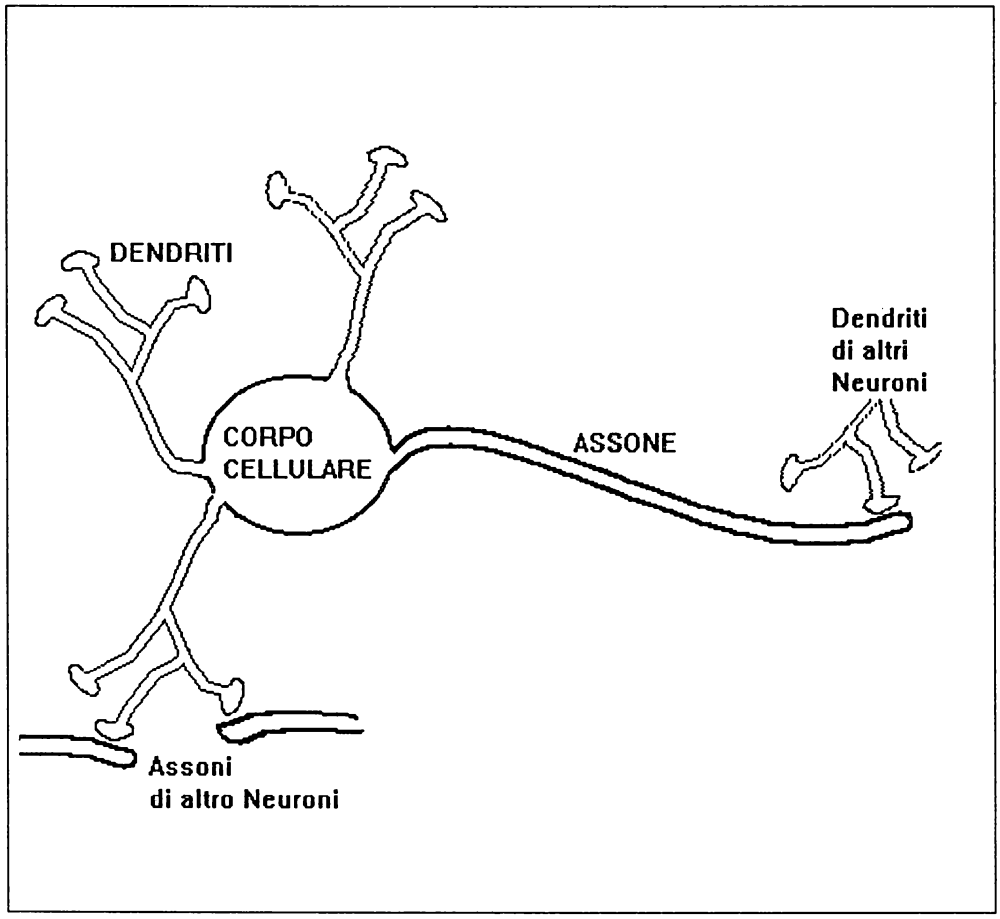

Figura 2. *La cellula neuronale*

Il corpo cellulare è rivestito da una membrana in grado di mantenere una polarizzazione, vale a dire una concentrazione di cariche elettriche. Tali cariche provengono dai dendriti, che sono dei prolungamenti ramificati a loro volta in contatto con altri neuroni. I molteplici dendriti di un neurone concorrono a eccitare il corpo cellulare il quale, oltre un certo limite, "scarica" la sua attivazione lungo l'assone, andando quindi a interessare altri neuroni. Il punto di contatto fra due neuroni si chiama *sinapsi* e si manifesta mediante l'avvicinamento fra l'assone di un neurone con il dendrite di un altro neurone (figura 3).

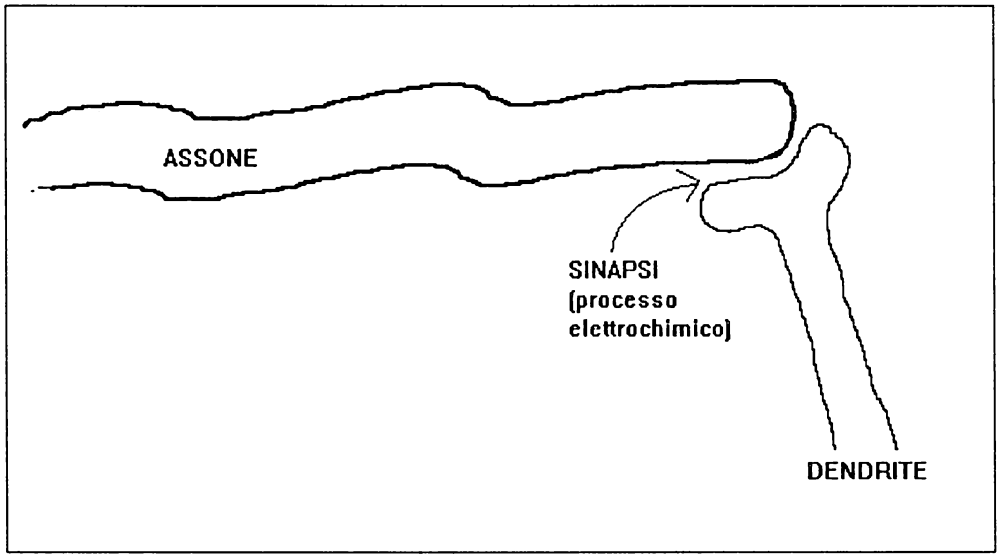

Figura 3. *La sinapsi.*

La sinapsi è una leggera intercapedine fra assone e dendrite in cui il segnale proveniente dall'assone si trasmette con un processo elettrochimico verso il dendrite. Lo spessore di quest'intercapedine è dell'ordine di poche decine di Angstrom e può variare nel tempo, provocando quindi un rafforzamento o indebolimento della connessione fra due neuroni.

Il contenuto informativo del cervello umano è rappresentato dall'insieme dei valori di attivazione di tutti i neuroni e l'elaborazione dell'informazione è rappresentata dal flusso di segnali fra i vari neuroni che si eccitano o inibiscono a vicenda. La memoria è rappresentata dall'insieme delle sinapsi che formano addensamenti e rarefazioni nella rete neurale [2].

La frequenza massima di trasmissione degli impulsi fra neuroni è piuttosto bassa, circa un centinaio al secondo [3], quindi l'informazione trasmessa è limitata e insufficiente per dati complessi. In un computer i segnali elettronici viaggiano a velocità migliaia di volte superiori. Ciò spiega l'ipotesi di un meccanismo *diffuso* e *parallelo* di trasmissione dell'informazione, secondo la quale l'elaborazione risiede in intere regioni del cervello piuttosto che in singole unità. Ad avvalorare questa tesi sono anche diversi esperimenti biologici in cui si è notato che, in seguito ad una lesione cerebrale, le prestazioni intellettuali degradano proporzionalmente alla superficie della lesione; al contrario, lesioni di limitata superficie non annullano del tutto le capacità intellettuali e provocano lo stesso effetto al variare della zona dove avviene la lesione [4].

Se poniamo a confronto il cervello biologico con un computer elettronico possiamo notare una grossa differenza strutturale: in un computer, infatti, l'elaborazione viene effettuata in maniera sequenziale nell'unità centrale di calcolo (CPU) e la memoria è concentrata in unità separate (RAM). Anche i programmi software rispecchiano questa struttura, separando i dati dall'algoritmo e considerando un flusso di controllo sequenziale. Nel cervello invece l'informazione è distribuita su una moltitudine di unità elementari, di per sè semplici ma operanti in parallelo; inoltre non vi è netta distinzione fra memoria e unità di elaborazione.

Ciò che vi può essere in comune fra cervello e computer è il concetto di input e output (figura 4): negli esseri viventi, infatti, l'input è costituito dalle terminazioni nervose localizzate negli organi di senso (occhi, naso, bocca, orecchie, pelle...) e l'output è costituito da neuroni residenti negli organi motori (muscoli, corde vocali, occhi...).

Risalgono a oltre mezzo secolo fa i primi tentativi di schematizzare con modelli matematici il cervello umano. Nel 1943 McCulloch e Pitts presentarono un lavoro sulle reti "neurologiche" [5] proponendo un modello formale di neurone. Gli studi vertevano essenzialmente nell'individuazione delle funzioni computabili dalle reti neurali e nell'equivalenza con le funzioni computate dalla macchina di Turing (il padre dell'odierno computer).

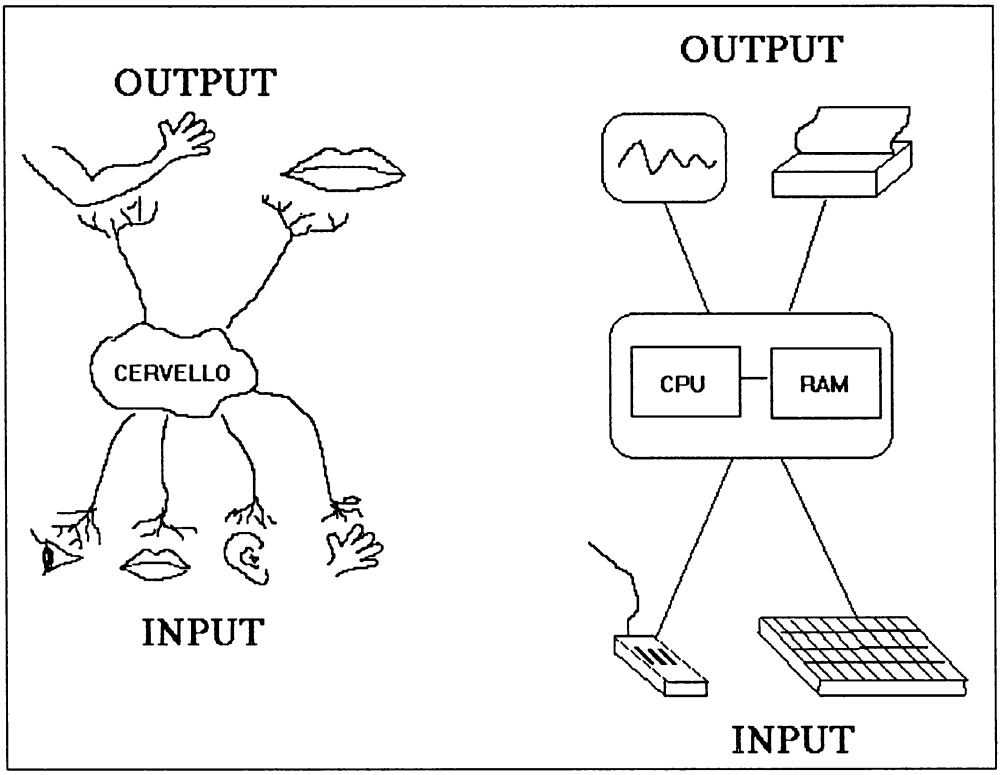

Figura 4. *Concetto di input/output negli esseri viventi e nel computer.*

Nel 1949 Donald Hebb ipotizzò l'apprendimento biologico come fenomeno sinaptico [6]: propose una formula matematica oggi nota come legge di Hebb. Cominciò a crescere l'entusiasmo verso questi dispositivi, che sembravano in grado non solo di svolgere svariate funzioni, ma anche di imparare da soli il compito loro richiesto.

Nel 1957 Rosenblatt costruì il cosiddetto *perceptrone* [7], un dispositivo in grado di imparare a riconoscere immagini: viene dimostrato un teorema che garantisce la capacità di apprendere un certo insieme di funzioni. Il perceptrone mostrava la robustezza e flessibilità tipica dei sistemi biologici: a differenza dei computer, un guasto o un errore nell'input non comprometteva l'intera elaborazione ma provocava un lieve degrado delle prestazioni. L'aspetto più sensazionale però stava nella capacità di riconoscere correttamente anche immagini mai viste prima, manifestando quindi capacità di *capire* piuttosto che di *imparare a memoria*, attraverso un meccanismo di associazione fra stimoli simili.

Nel 1969 Minsky e Papert analizzarono criticamente il perceptrone evidenziandone i limiti e l'incapacità di risolvere alcuni problemi banali, come ad esempio la funzione OR-esclusivo. Nel loro libro "Perceptrons" [8] manifestarono serie perplessità sulla possibilità di migliorare le reti neurali e ciò portò come conseguenza una sfiducia sull'argomento che dominò tutti gli anni 70.

Negli anni 80 si assiste ad una rinascita delle reti neurali in seguito al superamento dei limiti del perceptrone: gli studi di Kohonen [11] formalizzarono rigorosamente la materia e gli studi di Rumelhart [9] dimostrarono l'esistenza di reti dalla struttura piuttosto semplice in grado di svolgere funzioni complesse e significative con elevata accuratezza. Il trucco stava nell'impiego di neuroni "nascosti" che imparano dai propri errori (da qui il nome di questa tecnica: "error back propagation").

A questo si aggiungono i successi di Grossberg e quelli di Hopfield [10]. Quest'ultimo concepì le reti neurali non più come entità astratte (cioè governate da leggi matematiche arbitrarie), ma come entità fisiche (governate da leggi fisiche come quella dell'energia, della carica elettrica, dei flussi turbolenti...). Si è quindi dimostrato che le capacità intellettuali possono emergere come comportamento collettivo di un gran numero di entità fisiche di per sé stupide. Per la prima volta si è pensato di costruire reti neurali general-purpose senza l'aiuto del computer, ma utilizzando direttamente componenti elettronici.

Degni di nota infine i contributi di Kohonen [11] il quale ha identificato meccanismi di apprendimento per reti neurali che non richiedono l'esistenza di un maestro (unsupervised learning). In altre parole, si è dimostrata l'esistenza di reti neurali in grado di *scoprire* nuove conoscenze anziché *assimilare* le conoscenze altrui. È di particolare interesse l'architettura di queste reti, che sembra rispecchiare fedelmente le reti nervose biologiche.

Negli anni 80, i successi e le speranze dell'Intelligenza Artificiale simbolica (Sistemi Esperti, Linguaggi Logici, Reti Semantiche...) hanno alimentato una diatriba fra i sostenitori dei due diversi paradigmi per la rappresentazione di conoscenza: connessionista e simbolico. L'inizio degli anni 90 è caratterizzato da una forte tendenza ad integrare i due paradigmi.

Nel resto di questo libro andremo in dettaglio nella descrizione delle reti neurali senza ripercorrere le varie tappe storiche, ma presenteremo soltanto i modelli più significativi e consolidati di reti neurali in ordine crescente di difficoltà: dapprima il modello di Hopfield (memoria associativa), poi quello di Rumelhart (reti "backpropagation") e infine quello di Kohonen (reti auto-organizzanti).

2 Neuroni formali e reti artificiali

Il neurone formale è una schematizzazione del neurone biologico in cui le proprietà funzionali sono descritte da formule matematiche, senza preoccuparsi dei fenomeni elettrici, chimici, termici, biologici che avvengono nella realtà. Per distinguere il neurone formale da quello biologico, useremo una diversa terminologia: anziché neurone diremo semplicemente *unità*; anziche dendriti e assone diremo *input* e *output* dell'unità. Attenzione a non confondere l'input/output della singola unità con l'input/output dell'intera rete.

Lo stato di eccitazione di un neurone (polarizzazione elettrica) verrà rappresentato da un *valore di attivazione*, espresso come un numero reale. Il meccanismo di scarica dell'attivazione lungo l'assone verrà rappresentato da una *funzione di trasferimento* (o *funzione di output*) che darà il valore dell'output dell'unità a partire dal valore dell'attivazione. La sinapsi verrà chiamata *peso*, nel senso che quando due unità sono in stretto contatto diremo che la loro connessione ha un forte peso (in altre parole, gioca molto sul comportamento della rete).

Il funzionamento del neurone formale può essere così schematizzato. Data una rete formata da N unità, consideriamo l'i-esima unità (figura 5): questa riceve alcuni segnali di input provenienti dall'output di altre unità; indichiamo con O_j l'output della j-esima unità, espresso come numero reale. Il valore O_j viene trasmesso all'i-esima unità opportunamente pesato, cioè moltiplicato per il valore della forza sinaptica esistente fra la j-esima unità e l'i-esima: denoteremo tale peso con W_{ij} (dall'inglese Weight). Quindi il segnale che l'i-esima unità riceve sarà $W_{ij}O_j$. Si noti che se W_{ij} è compreso fra 0 e 1 avrà la funzio-

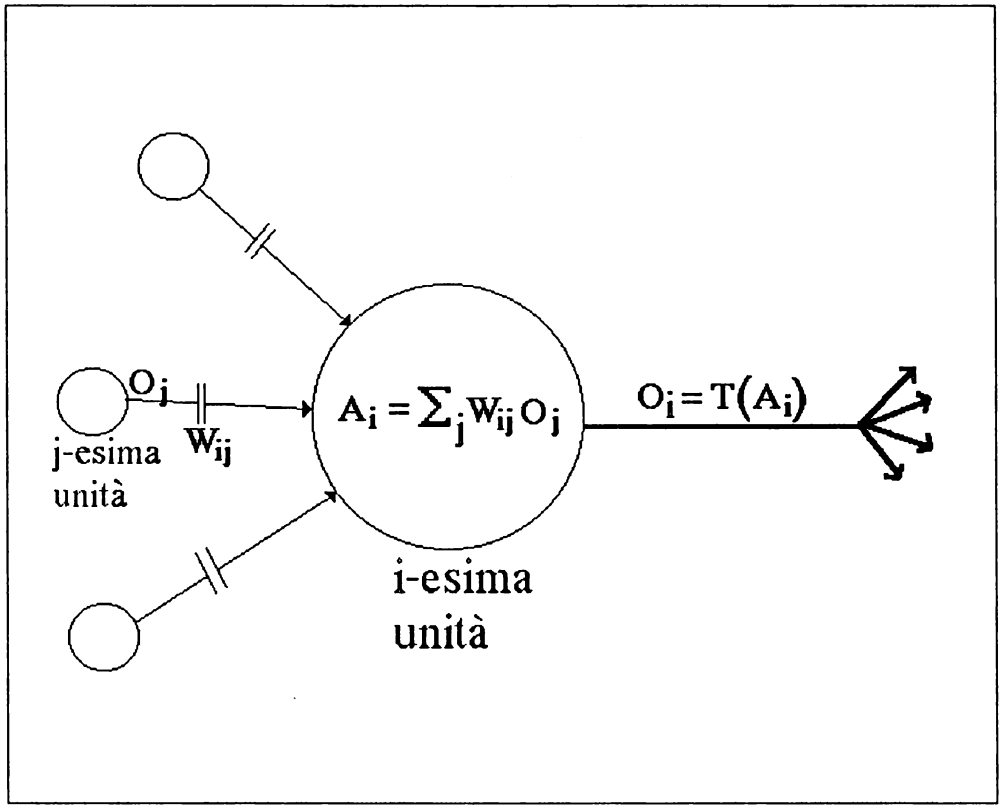

Figura 5. *Il neurone formale.*

ne di smorzare il segnale O_j; se è maggiore di 1 avrà una funzione amplificatrice; se è negativo avrà una funzione inibitrice anziche eccitatoria sull'unità i.

Le varie unità connesse con la i-esima unità concorrono a formare il suo stato di attivazione, che sarà dunque calcolato come sommatoria: indicando con A_i lo stato di attivazione dell'i-esima unità, avremo:

$$A_i = \sum_j W_{ij} O_j$$

Il valore dell'output dell'i-esima unità sarà dato dall'applicazione della funzione di trasferimento, denotata da T. Quindi:

$$O_i = T(A_i)$$

La funzione di trasferimento ha lo scopo di "trattenere" l'attivazione all'interno dell'unità fino ad un certo livello di soglia, oltre il quale l'unità si "scarica": la forma della funzione T sarà dunque a rampa (spigolosa o morbida, vedi figura 6).

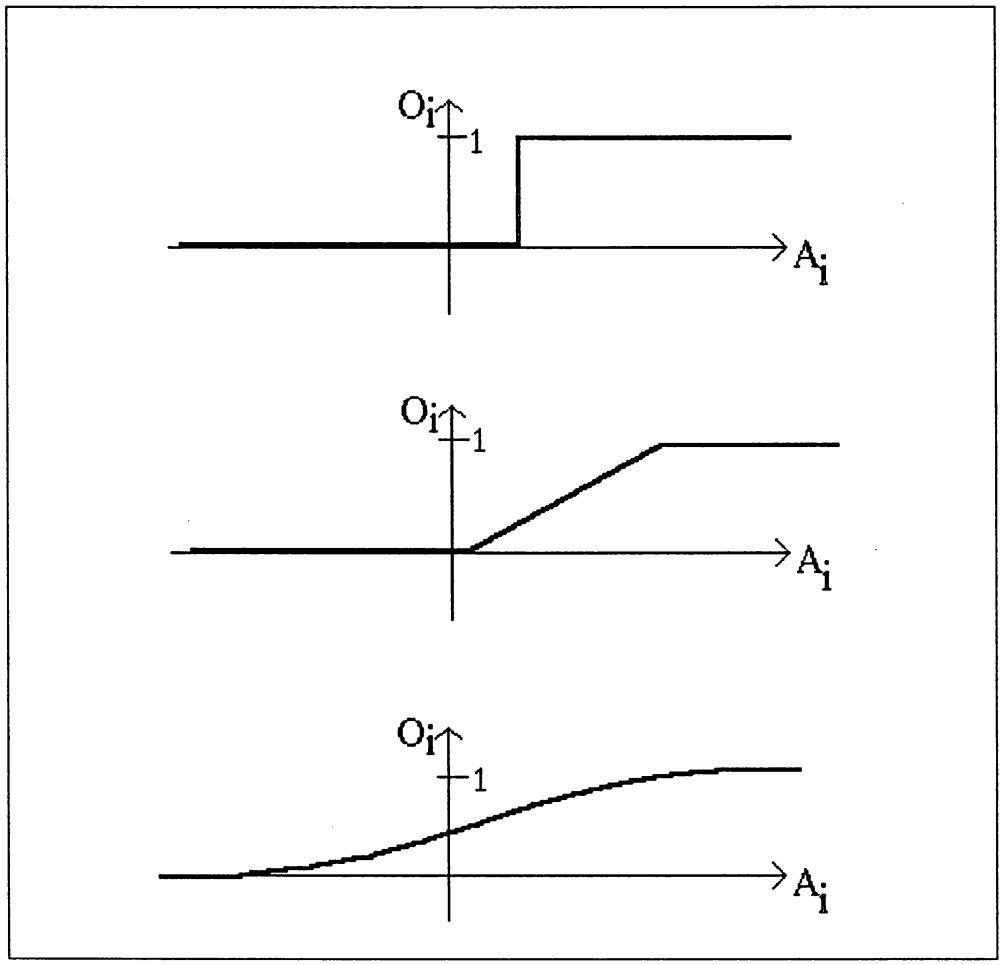

Figura 6. *Tipiche funzioni di trasferimento.*

Una rete neurale artificiale è formata da un certo numero di unità fra di loro interconnesse (figura 7). Alcune unità ricevono segnali dall'esterno e per questo verranno battezzate *unità di input* della rete; analogamente, le unità che forniscono il loro valore di output all'ambiente esterno saranno considerate *output* della rete (da non confondere con input/output del singolo neurone).

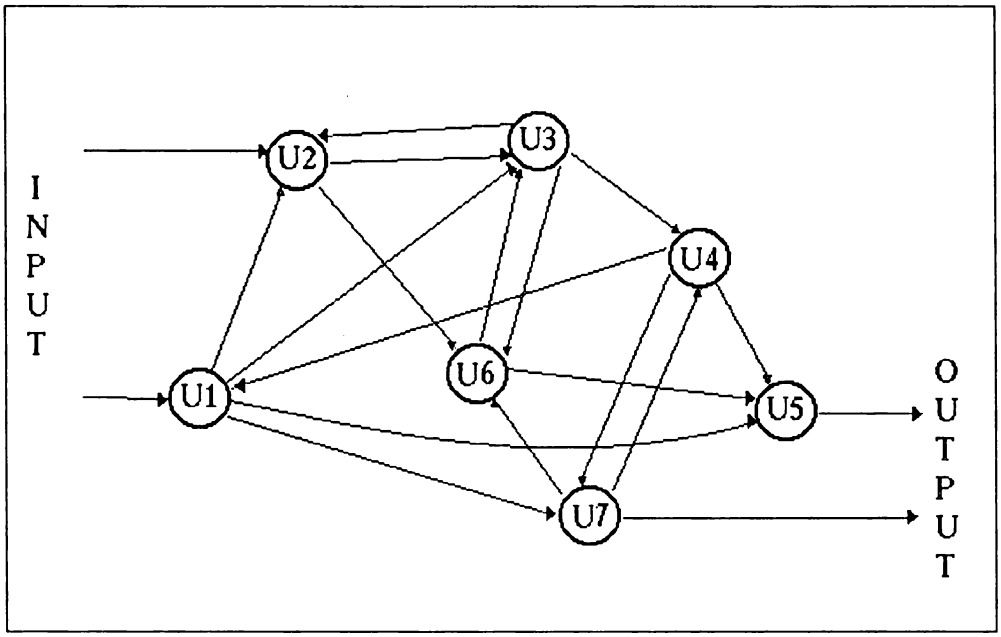

Figura 7. *Una rete neurale artificiale*

L'input di una rete neurale è dunque un insieme di numeri reali che indicheremo con X; analogamente indicheremo con Y l'insieme dei numeri che formano l'output della rete. Il compito della rete è quello di fornire un output ogni volta che gli viene presentato un input; in altre parole, una rete neurale calcola una funzione F (da non confondere con la funzione di trasferimento T):

$$Y = F(X)$$

L'esecuzione della funzione F avviene in questo modo: inizialmente la rete è in stato di quiete, quindi tutte le unità sono inattive e forniscono un output O_i nullo. Quando pervengono dall'esterno dei segnali di input X, le unità di input vengono sollecitate e quindi mandano un segnale alle altre unità; queste ultime ricevono uno stimolo che dipende dai pesi delle connessioni. Ogni unità della rete verrà dunque attivata in maniera diversa e propagherà stimoli alle altre unità. L'attivazione assunta dalle unità di output verrà trasmessa all'esterno formando l'insieme Y dei valori di output della rete. Si noti il fatto che l'elaborazione è avvenuta "in un sol colpo", attivando in modo parallelo tutte le unità simultaneamente.

Ma non è tutto così semplice: una rete neurale, infatti, può contenere connessioni cicliche, come ad esempio quelle in figura 7 fra le unità U2 e U3, oppure quelle che formano il ciclo U1-U3-U4. In questo caso alcune unità continuano a stimolarsi a vicenda e possono verificarsi tre situazioni:

1. la reciproca stimolazione porta ad attivare le unità di output con valori sempre differenti. In questo caso la rete fornirà all'ambiente un output in perenne cambiamento: diremo che la rete *non converge*. Può anche capitare che la stimolazione sia in continua crescita: in questo caso la rete *esplode*.
2. la reciproca stimolazione porta ad una sequenza di attivazioni che si ripetono in ciclo: in questo caso l'output *oscilla* in continuazione.
3. la reciproca stimolazione porta ad una combinazione di attivazioni che non cambia più: in questo caso diremo che la rete *converge* verso una soluzione Y=F(X). Questo è ovviamente il caso più interessante perché fa diventare la rete neurale una macchina deterministica: esistono teoremi che pongono le condizioni base affinché una rete sia sempre convergente.

Per facilitare lo studio dei fenomeni che avvengono in una rete neurale possiamo considerare alcuni modelli semplificati (figura 8):

1. Reti *non ricorrenti* = reti in cui le connessioni vanno in un solo senso, dall'input all'output. È il contrario di reti *cicliche*.
2. Reti *totalmente connesse* = reti in cui ogni unità è connessa con tutte le altre (generalmente esclusa sé stessa). Si noti che una connessione fra due unità con peso nullo, $W_{ij}=0$, è equivalente all'assenza di connessione: per comodità di calcolo si utilizzano spesso reti totalmente connesse in cui la topologia viene definita azzerando alcuni pesi.
3. Reti *a livelli* = reti i cui le unità sono organizzate in insiemi separati e disgiunti di cui generalmente uno è detto livello di input, un altro livello di output e gli altri livelli vengono chiamati *nascosti* o *intermedi*.
4. Reti *simmetriche* = reti in cui la connessione fra due qualsiasi unità è uguale in entrambi i sensi: $W_{ij}=W_{ji}$.

5. Reti *autoassociative* = reti in cui le unità di input coincidono con quelle di output. Il compito di queste reti è di ricevere uno stimolo dall'esterno e di farlo evolvere, fornendo come risultato una versione modificata (o completata) dell'input ricevuto.
6. Reti *stocastiche* = reti in cui vi è una certa probabilità che un'unità non venga attivata anche quando riceve stimoli.
7. Reti *asincrone* = reti in cui le unità vengono attivate non tutte contemporaneamente ma una alla volta secondo un ordine casuale.

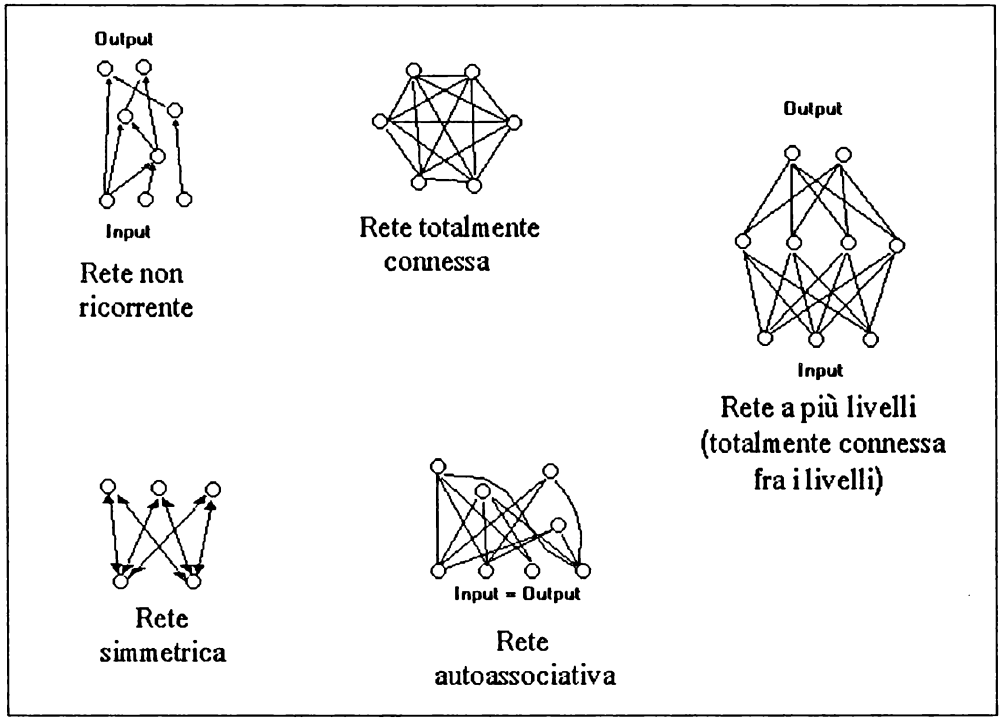

Figura 8. *Modelli di reti neurali*

Nei capitoli a seguire perlustreremo in dettaglio solo i modelli più interessanti e significativi: non faremo una rassegna completa.

3 Computabilità, apprendimento e caratteristiche

Sarebbe lecito domandarsi a questo punto: "ma cos'hanno queste reti neurali di diverso dai computer?" Tutto sommato anche le reti, come i computer, sono macchine che danno un certo output come frutto dell'elaborazione di un certo input (col termine "computer" intendiamo qui una macchina di Von Neumann, vale a dire sequenziale, programmabile e con memoria).

Prima di analizzare le differenze fra reti neurali e computer, soffermiamoci sugli aspetti che vi sono in comune. Esiste una tesi che afferma che la classe di funzioni computabili da una rete neurale coincide con la classe delle funzioni calcolabili su macchine di Turing. Tradotto in termini più terra-terra, questa tesi afferma che per ogni possibile programma per computer esiste una rete neurale in grado di fare le stesse cose.

Un altro risultato interessante è il teorema di Hecht-Nielsen [17]: questo afferma che una qualsiasi funzione Y=F(X) può essere computata con elevata accuratezza da una rete neurale *non ricorrente a soli tre livelli* avente un opportuno numero di unità (anche molto elevato) nel livello intermedio, senza alcuna connessione all'interno del livello ma totalmente connessa fra un livello e l'altro (figura 9).

Il teorema di Hecht-Nielsen riveste importanza più teorica che pratica, infatti non indica quante devono essere le unità né quali devono essere i pesi W per ottenere una rete che effettivamente esegua la funzione F. Non vi è una soluzione generale a questo

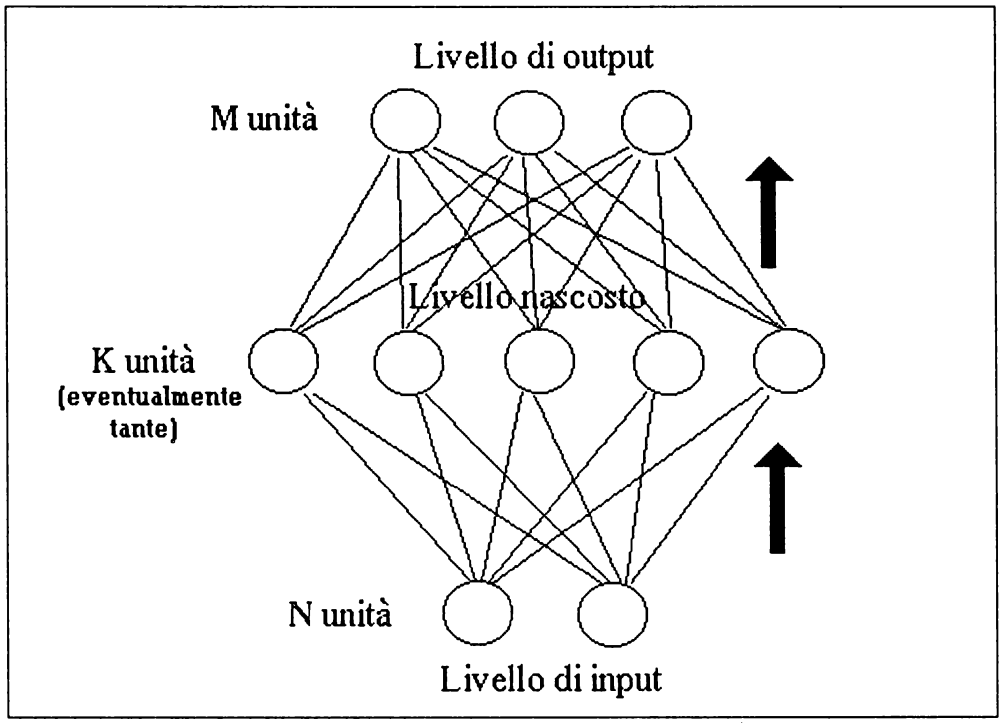

Figura 9. *Struttura di una rete in grado di computare una funzione qualsiasi*

problema, ma fortunatamente esistono tecniche che forniscono approssimazioni ben funzionanti nella pratica (fra cui la tecnica back propagation).

In conclusione, quindi con le reti neurali si riescono a fare essenzialmente le stesse cose che si possono fare con i computer. Però il fatto che l'informazione venga elaborata in maniera distribuita da una moltitudine di unità elementari, porta a due singolari conseguenze: *resistenza al rumore* e *resistenza al degrado.* Resistenza al rumore significa che la rete è in grado di operare anche in presenza di dati incerti, incompleti o leggermente errati. Resistenza al degrado significa che la rete è in grado di operare anche in presenza di guasti o parti difettose; ovviamente quest'ultima caratteristica ha un senso nel caso di reti neurali hardware realizzate direttamente con componenti elettronici (o biologici). Entrambe queste caratteristiche derivano dal fatto che l'informazione viene processata "a pezzettini" dalle varie unità della rete e l'elaborazione emerge come fenomeno collettivo; di conseguenza un difetto parziale nell'informazione, dovuto ad

errore d'uso o a guasto, affonda nella collettività e viene soprasseduto. Si noti l'estrema differenza rispetto ai computer tradizionali, in cui la presenza di "colli di bottiglia" (CPU, BUS, Flusso di controllo...) comporta una estrema sensibilità al minimo errore; ne consegue che le reti neurali sono appropriate in quei campi in cui più che la precisione è richiesta la robustezza: telecomunicazioni, visione, diagnostica, interpretazione di segnali con rumore (voce, caratteri manoscritti, firme), ...

Un'altra grossa differenza fra reti neurali e computer riguarda il meccanismo per ottenere le funzionalità desiderate. Stamo parlando, cioè, di come "programmare" una rete neurale. Una rete neurale non va programmata bensì *addestrata.*

Addestrare una rete neurale significa presentarle un insieme di esempi e lasciare che la rete si costruisca da sola la conoscenza interna necessaria per svolgere il compito richiesto. Un esempio (volutamente esagerato anche se realmente funzionante) è quello di insegnare ad una rete il concetto di somma: anziché fornirle un algoritmo (ad esempio somma le singole cifre da destra a sinistra con riporto), si presentano alla rete alcuni esempi di somme: 5+3=8; 24+1329=1353; 1+17=18... Dopo aver "osservato" un numero sufficiente di somme, la rete avrà imparato il concetto di somma, quindi sarà in grado di eseguire anche somme di numeri mai visti prima, come ad esempio 37+9.

Un altro esempio può essere quello di addestrare una rete ad effettuare dei movimenti fisici come ad esempio giocare a tennis o sciare. Per non complicarci troppo la vita, consideriamo un movimento elementare ma significativo come ad esempio quello per lanciare in aria un sasso per colpire un bersaglio. È sufficiente presentare alla rete un certo insieme di esempi, facendole vedere dove arrivano alcuni sassi lanciati per prova: analizzando questi tiri-prova, la rete riesce ad autocostruirsi la capacità di tirare sassi, tale per cui, quando le si chiederà di lanciare un sasso fino a laggiù, la rete saprà quale forza e quale angolazione imprimere al lancio.

Nei computer tradizionali, invece, questo problema deve essere risolto a mano dall'uomo, che deve trovare le equazioni del moto di un corpo in caduta libera e tradurre tali equazioni in un algoritmo da far eseguire dal computer. Il fatto che le reti neurali possano apprendere da esempi le rende particolarmente

appropriate in quei campi applicativi in cui non si riesca o non sia conveniente trovare una soluzione algebrica o algoritmica: si pensi ad esempio alle previsioni metereologiche, al riconoscimento di immagini, alla diagnostica medica, alla guida di autoveicoli, al controllo di parti meccaniche in movimento...

Si consideri il problema di riconoscere volti umani (ad esempio per consentire l'accesso ad un reparto privato). Purtroppo non esiste una soluzione algoritmica a questo problema, perché ogni volto è un caso a parte. Un possibile approccio può essere l'impiego di una rete neurale che *autoapprenda* a riconoscrere i volti "amici", mediante l'analisi di fotografie. Dato un opportuno insieme di foto-esempio, la rete imparerà a riconoscere persone anche con diversa pettinatura o sorridenti o col cappello... Il "trucco" sta nel fatto che la rete impara a distinguere i tratti che sono tipici di un volto (occhi, orecchie, naso...) da quelli insignificanti (espressione, pettinatura, trucco...).

Per correttezza è bene puntualizzare che le capacità di apprendimento di una rete neurale (come anche la resistenza al rumore e al degrado) non sono frutto di magia, ma di una accurata scelta della topologia, numero di unità, regola di apprendimento, insieme di esempi... In altre parole, può capitare (ed è frequente) che una rete non riesca affatto ad apprendere la funzione desiderata oppure che non resista al rumore, semplicemente perché abbiamo usato un'architettura non adeguata (per esempio carenza di unità o di livelli, uso di una funzione di trasferimento non adeguata...).

Da un punto di vista biologico per apprendimento si intende la capacità del sistema nervoso degli esseri viventi di autoadattarsi a fronte di stimoli esterni, modificando la propria conoscenza. Gli studi di neurofisiologia hanno portato all'ipotesi che la capacità di autoadattamento risieda in una plasticità sinaptica, vale a dire variazione della forza di connessione fra le cellule nervose.

In termini matematici, addestrare una rete significa fornirle un insieme di coppie input/output (X,Y) e fare in modo che la rete trovi da sola i valori delle connessioni W che realizzino la funzione Y=F(X). Riassumendo:

apprendimento = trovare W dato X e Y;

esecuzione = trovare Y dato X e W.

L'idea base dell'apprendimento neurale è la seguente: per ogni esempio presentato, rafforzare le connessioni fra unità che siano entrambe attive. Più propriamente, durante la presentazione di un certo esempio, incrementare W_{ij} proporzionalmente a quanto sono attive le unità i e j. Da un punto di vista matematico, questa regola è espressa dalla cosiddetta formula di Hebb [6]:

$$\Delta W_{ij} = \varepsilon O_i O_j$$

Si noti che quanto più sono alti i valori delle unità i e j, tanto più è alta la modifica del peso fra queste due unità; il tutto rapportato da una costante di proporzionalità ε, comunemente chiamata *tasso di apprendimento.*

Questa formula, pur funzionando in diverse situazioni pratiche, non è del tutto fatata, infatti non garantisce la capacità di apprendere qualsiasi funzione; in altre parole, questa regola ha due difetti: modifica immancabilmente i pesi anche quando non necessario e, al contrario, non modifica abbastanza i pesi nei casi più ardui. Ecco dunque la necessità di cambiare formula pur tenendo valido il principio di base. Il trucco è quello di applicare a ripetizione una formula che "aggiusti" per piccoli passi ogni peso dirigendolo verso il valore ottimale. Ad ogni passo è necessario lasciare che la rete evolva da sola per vedere quanto ha sbagliato, quindi modificare solo i pesi delle unità che hanno errato di un ammontare tale da compensare l'errore commesso. Il tutto ripetuto finché non vi sono pesi errati, o meglio, finché l'errore commesso non diventa accettabile. In conclusione, ciclando in continuazione sullo stesso esempio, o sullo stesso insieme di esempi, otteniamo miglioramenti graduali che portano alla combinazione ottimale di pesi. Ognuno di questi cicli, in cui vengono ripresentati sempre gli stessi esempi, prende il nome di *epoca* di apprendimento. La formula dell'apprendimento per epoche è dunque:

$$\Delta W_{ij} = \varepsilon\left(D_i - O_i\right)O_j$$

dove D_i indica il valore desiderato di O_i, quindi D_i-O_i indica l'errore commesso dall'unità i.

Attenzione: la fase di apprendimento di una rete è ben distinta da quella di esecuzione; una rete o impara o esegue un certo compito, analogamente ai computer tradizionali in cui la fase di programmazione è ben distinta da quella di esecuzione. Da questo punto di vista le reti neurali artificiali differiscono da quelle

biologiche, in cui l'apprendimento è sempre "acceso". Si pensi ad attività sportive complesse, come ad esempio sciare: anche i maestri più anziani sono in continuo allenamento e non potranno mai sciare in modo totalmente meccanico. Una rete neurale artificiale, invece, richiede una fase di addestramento intensivo, dopo la quale viene "commutata" in modalità di esecuzione, in cui non impara più (pur rimanendo valida la possibilità di "ricommutarla" in modalità di apprendimento per aggiornare la sua conoscenza).

L'apprendimento da esempi di una rete neurale porta implicitamente ad un'altra straordinaria capacità: quella di *generalizzazione*. L'addestramento di una rete neurale può avvenire senza doverle fornire tutti gli esempi possibili immaginabili, ma soltanto i più significativi e rappresentativi. Ad esempio, ritornando ai tiri del sasso, non è necessario presentare alla rete tutti i possibili tiri-prova (che peraltro sarebbero infiniti), ma soltanto una piccola parte, ad esempio 200 tiri di cui un pò vicini, un pò lontani, alcuni alti e alcuni tesi. Supponiamo che fra questi esempi fossero presenti un tiro a 12 metri bello alto e uno a 15 metri piuttosto teso, ma non vi fosse alcun altro tiro fra i 12 e i 15 metri. La capacità di generalizzazione della rete la rende capace di raccapezzarsi anche nel caso di un tiro a 13 metri molto teso, indovinando per analogia la giusta combinazione fra angolazione del lancio e forza del tiro.

In conclusione, quindi una rete neurale manifesta capacità di *comprensione* piuttosto che di *memorizzazione*; si noti la differenza rispetto ad un tradizionale sistema di database il quale, una volta memorizzati i suoi dati è in grado solo di ripescarli così come sono senza fare analogie e somiglianze per potersi districare di fronte a dati non memorizzati o situazioni nuove. La capacità di generalizzazione di ciò che si apprende è una caratteristica tipica dell'intelligenza umana: sarà capitato anche a voi, ad esempio, di aver preso un brutto voto a scuola pur avendo imparato a memoria tutto il capitolo di storia o geografia. Magari prendendovi pure dello stupido! Al contrario, uno studio poco mnemonico ma ben fornito di analogie, paragoni e astrazioni porta immancabilmente a buoni risultati perché è indice di comprensione e inquadramento piuttosto che di bruta memorizzazione (diamo per scontato che nella vita di tutti i giorni è più utile capire che imparare a memoria).

Esistono due principali regimi di apprendimento: *con supervisione* e *senza supervisione* (supervised e unsupervised). Gli esempi citati in precedenza (somma di numeri, lancio di sassi, riconoscimento di volti) erano tutti relativi ad apprendimento con supervisione, cioè caratterizzati dal fatto che gli esempi mostrati al discente specificano sia l'input che l'output.

L'apprendimento senza supervisione invece si manifesta quando gli esempi osservati contengono solo l'input. In questo caso non si tratta più di apprendere una funzione Y=F(X), ma di organizzare e inquadrare gli esempi osservati trovando relazioni e regolarità fra di loro. Facciamo subito un esempio: supponiamo di avere un insieme di dati relativi ad aziende italiane, in cui per ogni azienda siano citate le dimensioni, l'ubicazione, il tipo di prodotto, il fatturato... insomma, un tradizionale database. Considerando ogni azienda come un esempio, una rete neurale con apprendimento senza supervisione è in grado di individuare raggruppamenti di aziende simili, arrivando a *scoprire* concetti come aziende private, aziende di servizi, aziende del nord... oppure *notare* relazioni come ad esempio "le compagnie aeree sono generalmente statali", "la tipica azienda del nord è piccola e privata", "le aziende di servizi hanno un basso fatturato pro capite"...

Terminiamo questo capitolo con un riassunto delle principali caratteristiche delle reti neurali.

1. Una rete neurale è un sistema dinamico in grado di eseguire qualsiasi funzione Y=F(X).
2. La struttura di una rete neurale è definita dal numero di unità, dalla particolare topologia (a livelli, ciclica, totalmente connessa...) e dal tipo di funzione di trasferimento T.
3. Una rete neurale non la si programma ma la si addestra. L'addestramento richiede un insieme di esempi specifici anziché una regola o un algoritmo generale. L'apprendimento può essere con o senza supervisione ed è espresso nella forma $\Delta W_{ij} = ...$
4. La "conoscenza" di una rete neurale (memoria a lungo termine, programma) risiede nell'insieme dei pesi delle connessioni fra le unità e nell'insieme di soglie di attivazione delle unità. Tale conoscenza non è in forma simbolica espli-

cita ma è un guazzabuglio di numeri di difficile comprensione e interpretazione da parte umana.

5. I "dati" di una rete neurale (vale a dire le informazioni sul singolo caso in esame, memoria a breve termine) risiedono nei valori di attivazione che le unità assumono in seguito ad uno stimolo di input.
6. Una rete neurale non ha un orologio centrale né un flusso di controllo; l'elaborazione è frutto di un fenomeno distribuito e collettivo, che può essere eseguito con alto parallelismo ed eventuale asincronia.
7. Non vi è distinzione fra hardware e software nelle reti neurali: sono concepibili sia come macchine a sé stanti costruite ad hoc, sia come modelli matematici simulati via software sugli attuali computer (sequenzializzando l'aspetto parallelo).
8. Punti deboli delle reti neurali: scarse capacità di calcolo, scarse capacità logiche, scarsa precisione, incapacità di spiegazione dei risultati forniti.
9. Punti forti delle reti neurali: capacità di associazione (fare analogie, trovare similarità), capacità di generalizzazione, resistenza al rumore e al degrado, familiarità con gli eventi più frequenti, velocità (su hardware parallelo).
10. Una rete neurale non si misura in byte o in istruzioni al secondo ma in unità e connessioni al secondo.

4 Memoria associativa distribuita

Col termine "memoria associativa" si intende un dispositivo in grado di contenere informazioni sotto forma di associazione fra dati. Un sinonimo di memoria associativa può essere *memoria accessibile per contenuto*; il contrario è *memoria accessibile per indirizzo*. Un esempio tipico può essere l'associazione fra il nome dei miei amici e il mestiere che fanno: se ad esempio Ruggero fa lo skipper, il suo nome mi verrà in mente ogni qual volta si parla di vela e, viceversa, mi verranno in mente barche a vela ogni volta che si parla di Ruggero.

Al contrario, una tipica memoria accessibile per indirizzo è un'agenda, in cui è estremamente facile sapere cosa ho fatto il 19 Aprile (usando la data come indirizzo d'accesso) ma è altamente oneroso sapere quando ho incontrato Jacopo, infatti dovrei scandire tutta l'agenda verificando giorno per giorno gli incontri fatti. Un memoria associativa, dunque, effettua la classica *associazione di idee*, spesso caratterizzata dal fatto che un'informazione "fa venire in mente" un'altra informazione per analogia e similitudine piuttosto che per uguaglianza esatta.

Una buona memoria associativa comporta l'abilità di operare in presenza di dati incompleti o errati. Si supponga ad esempio di immagazzinare in una memoria associativa un catalogo di libri, contenente la voce "Emilio Salgari: Le tigri di Mompracem, scaffale 21". L'associazione fra autore e titolo permetterà di reperire un dato per intero anche a partire da una porzione di esso, come

ad esempio "Salgari: Le tigri"; oppure la possibilità di *correggere* versioni distorte di questo dato, come ad esempio "Egidio Salgari: Le tigri di Montparcel". È infine auspicabile la capacità di accedere alle informazioni memorizzate con prestazioni indipendenti dalla quantità di dati immagazzinati; in altre parole, il tempo di accesso ad una informazione deve essere lo stesso sia in una base dati di poche righe che di migliaia. Questa caratteristica è tipica degli esseri viventi, infatti l'abilità di riconoscere un volto, una musica, un'odore è identica a 5 anni come a 50, però a 50 anni le informazioni immagazzinate nel cervello sono assai di più che a 5 anni. Questo suggerisce l'ipotesi di un supporto di memoria che non comporti l'elencazione sequenziale dei dati, come avviene negli attuali database, ma la *sovraimposizione* dei dati in una stessa struttura.

Una rete neurale è particolarmente adatta per realizzare memorie associative perché l'informazione risiede nell'insieme delle connessioni fra le varie unità e il singolo dato da memorizzare non è "concentrato" in celle di memoria ma è *distribuito* su una moltitudine di unità; da qui il nome *memoria associativa distribuita.* Le unità della rete non sono specializzate per la memorizzazione di un particolare dato, ma offrono ciascuna un piccolo contributo alla memorizzazione di tutti i dati.

Ma veniamo al dunque: qual è la struttura di una rete neurale che realizzi una memoria associativa? Ne esistono diverse, ma noi considereremo qui una delle più semplici, chiamata "memoria associativa bidirezionale" (BAM, Kosko [14]): due livelli di unità, che chiameremo livello X e Y, con connessioni totali fra i due livelli e nessuna connessione all'interno del livello. Questo significa che ogni unità del livello X è connessa con tutte le unità di Y ma con nessun'altra unità di X (figura 10)

Indichiamo con N il numero di unità del livello X e con M quello del livello Y; ogni unità può assumere due soli valori: +1 e -1 e per questo vengono dette bipolari. La scelta di {+1,-1} anziché {0,1} (unità binarie) è dovuta al fatto che questo facilita le formule matematiche e migliora le capacità di memoria della rete. Le connessioni fra le unità sono bidirezionali e simmetriche, cioè $W_{ij}=W_{ji}$. La funzione di trasferimento T è un semplice gradino centrato sullo zero (figura 11), cioè la *soglia* oltre la quale l'unità "scatta" è il valore di attivazione 0. Per comodità, anziché indicare l'output delle unità con O_i, distingueremo l'output delle unità del

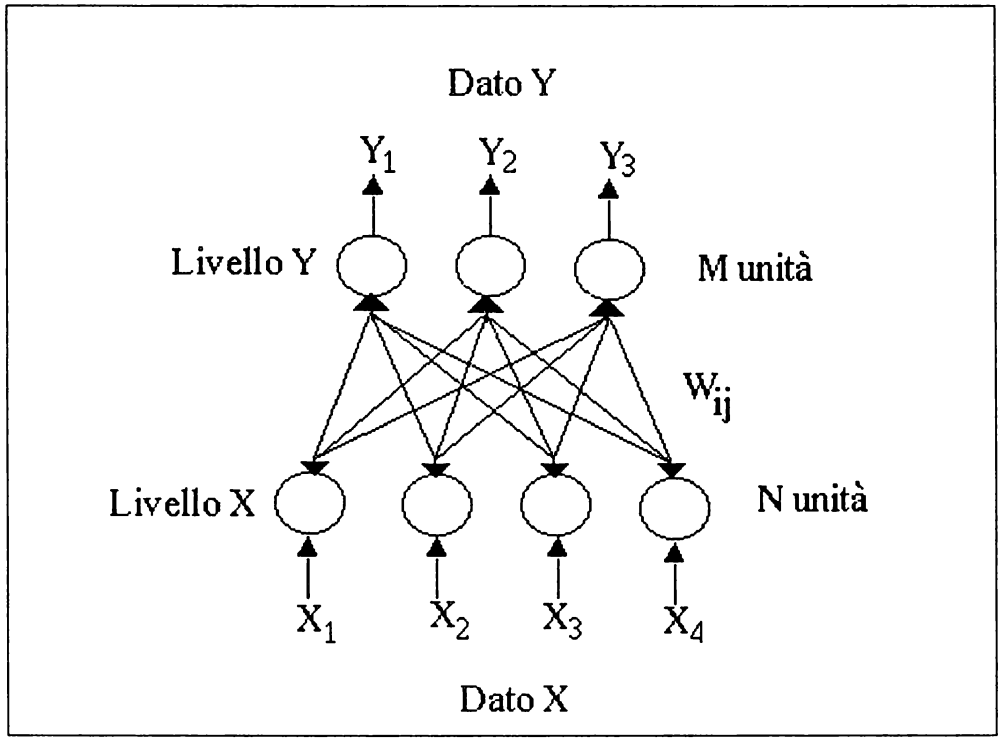

Figura 10. *Struttura di una rete neurale che realizza una memoria associativa distribuita.*

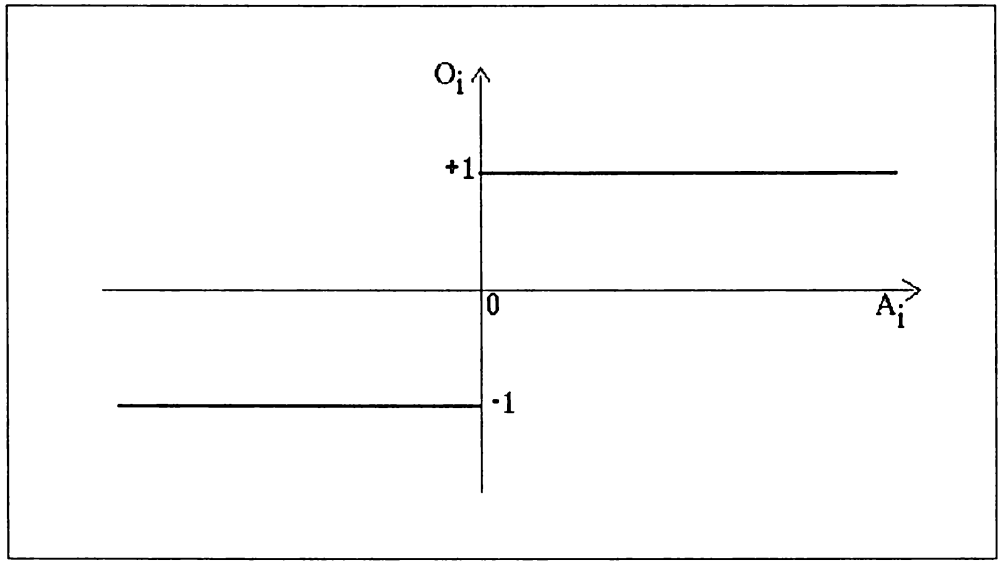

Figura 11. *La funzione di trasferimento per la memoria associativa.*

livello X da quelle delle unità del livello Y, denotandole rispettivamente X_i e Y_j.

Le formule che governano la rete sono dunque:

$$X_i = \begin{cases} +1 \; se \; \sum_j W_{ij} Y_j > 0 \\ -1 \; se \; \sum_j W_{ij} Y_j \leq 0 \end{cases}$$

$$Y_j = \begin{cases} +1 \; se \; \sum_i W_{ij} X_i > 0 \\ -1 \; se \; \sum_i W_{ij} X_j \leq 0 \end{cases}$$

La formula di apprendimento di una rete di questo tipo può essere la semplice formula di Hebb:

$$\Delta W_{ij} = X_i Y_j$$

La memorizzazione di un dato, o meglio, dell'associazione fra due dati X e Y, avviene semplicemente rappresentando i dati in formato bipolare (cioè con +1 e -1), assegandoli alle unita dei livelli X e Y e infine applicando la formula di Hebb (in letteratura si parla spesso di "pattern" anziché di "dati", per evidenziare il fatto che ogni dato è una combinazione di +1 e -1). La memorizzazione di più associazioni avviene ripetendo questo procedimento diverse volte con diversi dati.

Si supponga di avere 6 unità nel livello X e 4 nel livello Y; l'associazione fra due dati verrà espressa ad esempio in questo modo:

```
{ X1=(+1 -1 +1 -1 +1 -1), Y1=(+1 +1 -1 -1) }
```

Le connessioni fra le unità saranno 6x4=24 e possono essere rappresentate con una matrice di numeri interi di 6 righe e 4 colonne. Proviamo a memorizzare nella rete i dati X1 e Y1 visti sopra (in altre parole, addestriamo la rete con quell'esempio).

Dalla formula di Hebb otteniamo una matrice così fatta:

```
+1 +1 -1 -1
-1 -1 +1 +1
+1 +1 -1 -1
-1 -1 +1 +1
+1 +1 -1 -1
-1 -1 +1 +1
```

In questa matrice l'i-esima riga rappresenta i valori delle 4 connessioni che vanno dall'i-esima unità del livello X alle 4 unità del livello Y (figura 12).

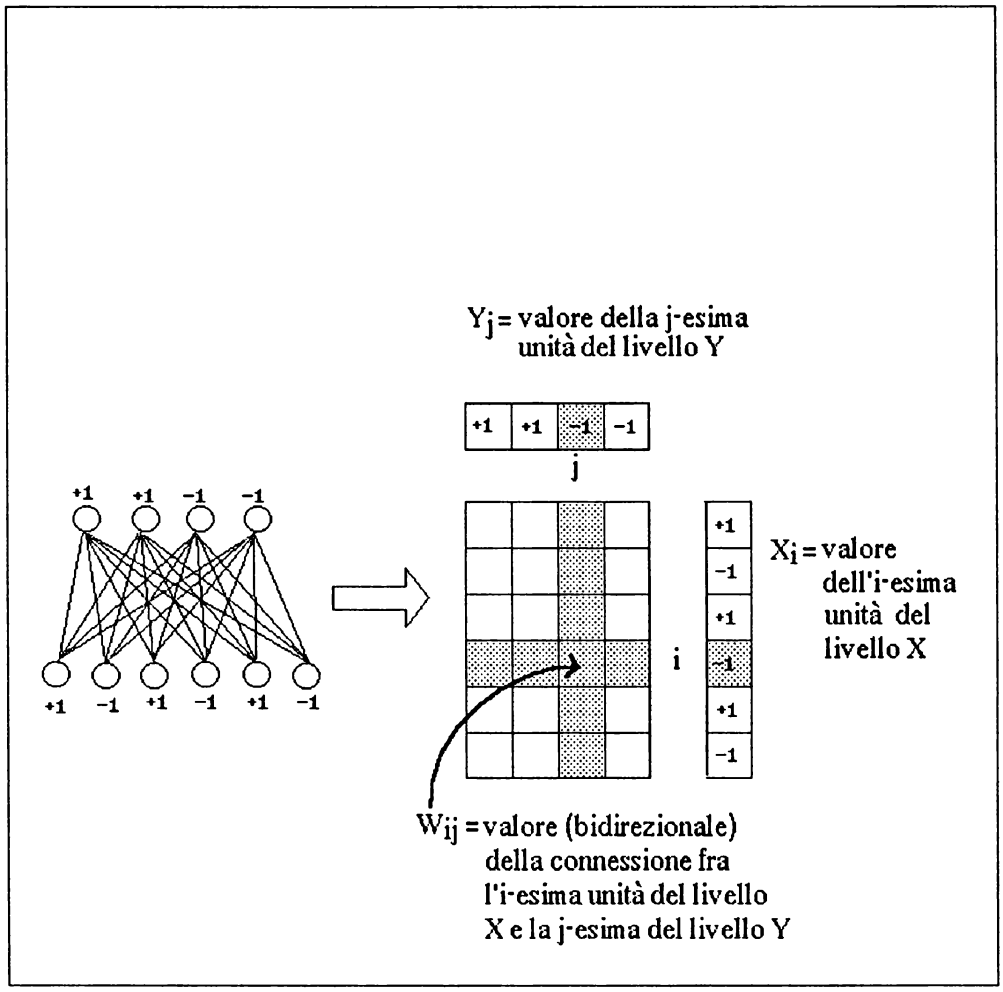

Figura 12. *Implementazione di una rete neurale con strutture dati per computer.*

Proviamo ora ad addestrare questa rete con un'altra associazione:

```
{ X2=(+1 +1 +1 -1 -1 -1), Y2=(+1 -1 +1 -1) }
```

La formula di Hebb implica che ogni connessione W_{ij} venga modificata *sovraimponendovi* il nuovo valore, ottenendo quindi:

```
+2  0  0 -2
 0 -2 +2  0
+2  0  0 -2
-2  0  0 +2
 0 +2 -2  0
-2  0  0 +2
```

In conclusione, un algoritmo per simulare questa rete neurale in modo sequenziale su un computer può essere:

```
X:array[1..N] of integer;
Y:array[1..M] of integer;
W:array[1..N, 1..M] of integer;
 procedure addestra;
  acquisisci dato X;
  acquisisci dato Y;
  per ogni unità j del livello Y
    per ogni unità i del livello X
      W[i,j] = W[i,j] + ( X[i] * Y[j] )
    fine
  fine
fine;
```

Fermiamo qui la fase di addestramento e passiamo a far eseguire questa rete, vale a dire assegnamo solo uno dei due livelli, diciamo X, e lasciamo evolvere la rete fino a quando si rilassa in uno stato stabile; a questo punto l'altro livello, Y, conterrà l'informazione associata ad X. Si noti che la rete è fortemente ciclica, infatti essendo le connessioni bidirezionali, ogni unità influenza le altre, a cui ne è a sua volta influenzata. Fortunatamente esiste un teorema che assicura la convergenza della rete garantendo l'impossibilità di oscillare per sempre caoticamente [15].

L'algoritmo per simulare in modo sequenziale l'esecuzione della rete è il seguente:

```
procedure esegui;
  acquisisci dato X;
  azzera dato Y;
  ripeti
```

```
    per ogni unità j del livello Y     (* andata da X a Y *)
      calcola la sua attivazione A=SUMi(W[i,j]*X[i]);
      Y[j]=T(A);
    fine;
    per ogni unità i del livello X     (* ritorno da Y a X *)
      calcola la sua attivazione A=SUMj(W[i,j]*Y[j]);
      X[i]=T(A);
    fine;
    visualizza dato Y;
  finché non ci sono più cambiamenti in X e in Y
fine;

funzione T(A);
  se A>0 allora T:=+1 altrimenti T:=-1;
fine;
```

Come si può notare, l'evoluzione della rete è una continua oscillazione da X a Y (andata e ritorno), fino a che non vi sono più cambiamenti (questo equivale ad una oscillazione infinita in cui da un certo punto in poi non cambia più nulla).

Vediamo quindi se la rete ha memorizzato correttamente i dati fornitele in fase di addestramento: proviamo a darle X=(+1 +1 +1 -1 -1 -1) e calcoliamo a mano il valore di Y, supponendolo inizialmente nullo, cioè Y=(0 0 0 0).

```
Y[1] = T(+1*2 +1*0 +1*2 -1*-2 -1*0 -1*-2) = T( 2+0+2+2-0+2)
                                = T(+8) = +1
Y[2] = T(+1*0 +1*-2 +1*0 -1*0 -1*2 -1*0)  = T( 0-2+0-0-2-0)
                                = T(-4) = -1
Y[3] = T(+1*0 +1*2 +1*0 -1*0 -1*-2 -1*0)  = T( 0+2+0-0+2-0)
                                = T(+4) = +1
Y[4] = T(+1*-2 +1*0 +1*-2 -1*2 -1*0 -1*2) = T(-2+0-2-2-0-2)
                                = T(-8) = -1
```

Come si può vedere, risulta Y=(+1 -1 +1 -1); calcoliamo ora il "ritorno" da Y a X:

```
X[1] = T(+1*2 -1*0 +1*0 -1*-2) = T( 2-0+0+2) = T(+4) = +1
X[2] = T(+1*0 -1*-2 +1*2 -1*0) = T( 0+2+2-0) = T(+4) = +1
X[3] = T(+1*2 -1*0 +1*0 -1*-2) = T( 2-0+0+2) = T(+4) = +1
X[4] = T(+1*-2 -1*0 +1*0 -1*2) = T(-2-0+0-2) = T(-4) = -1
X[5] = T(+1*0 -1*2 +1*-2 -1*0) = T( 0-2-2-0) = T(-4) = -1
X[6] = T(+1*-2 -1*0 +1*0 -1*2) = T(-2-0+0-2) = T(-4) = -1
```

e otteniamo X=(+1 +1 +1 -1 -1 -1), che è identico al valore precedente, quindi è inutile proseguire con altre oscillazioni, perché la

rete è ormai in uno stato stabile da cui non si discosta più. In conclusione, il dato Y fornito dalla rete come risposta allo stimolo iniziale X è stato esattamente quello che aveva appreso.

Proviamo ora a farla funzionare al contrario: diamo alla rete Y=(+1 -1 +1 -1) e vediamo cosa ci dà in risposta. La "andata" da Y a X è uguale a quella appena vista sopra mentre il "ritorno" da X a Y è uguale a quello visto prima. Anche in quasto caso fin dalla prima oscillazione la rete fornisce la corretta risposta. Proviamo ora con l'altro dato appreso, X=(+1 -1 +1 -1 +1 -1):

```
Y[1] = T(+1*2 -1*0 +1*2 -1*-2 +1*0 -1*-2) = T( 2-0+2+2+0+2)
                                = T(+8) = +1
Y[2] = T(+1*0 -1*-2 +1*0 -1*0 +1*2 -1*0)  = T( 0+2+0-0+2-0)
                                = T(+4) = +1
Y[3] = T(+1*0 -1*2 +1*0 -1*0 +1*-2 -1*0)  = T( 0-2+0-0-2-0)
                                = T(-4) = -1
Y[4] = T(+1*-2 -1*0 +1*-2 -1*2 +1*0 -1*2) = T(-2-0-2-2+0-2)
                                = T(-8) = -1
```

Come previsto, risulta Y=(+1 +1 -1 -1) al primo colpo. Il ritorno lo lasciamo calcolare a voi come esercizio contro l'insonnia. In conclusione, questa rete ha memorizzato correttamente le associazioni {X1,Y1} e {X2,Y2}. Forse l'impressione è di aver sparato ad una mosca con un cannone, andando a scomodare le reti neurali per fare un banale sistema che memorizza dati. Però il bello ha ancora da venire. Immaginiamo una rete con migliaia di unità nei livelli X e Y e migliaia di associazioni memorizzate: facendo andare la rete in parallelo (su hardware apposito) otteniamo il reperimento di una informazione in un batter d'occhio, senza dover scandire alcuna struttura sequenziale. Tutto ciò grazie alla sovraimposizione dei diversi dati nella stessa struttura e l'indipendenza fra le unità, che permette una elaborazione parallela.

Ma non è tutto: questa rete è resistente al rumore e al degrado. Proviamo a fornirle un input molto simile a X2 ma con un piccolo errore (nell'ultima unità): X=(+1 +1 +1 -1 -1 +1). Otteniamo (si noti l'ultimo addendo):

```
Y[1] = T(+1*2 +1*0 +1*2 -1*-2 -1*0 +1*-2) = T( 2+0+2+2-0-2)
                                = T(+4) = +1
Y[2] = T(+1*0 +1*-2 +1*0 -1*0 -1*2 +1*0)  = T( 0-2+0-0-2+0)
                                = T(-4) = -1
Y[3] = T(+1*0 +1*2 +1*0 -1*0 -1*-2 +1*0)  = T( 0+2+0-0+2+0)
                                = T(+4) = +1
Y[4] = T(+1*-2 +1*0 +1*-2 -1*2 -1*0 +1*2) = T(-2+0-2-2-0+2)
                                = T(-4) = -1
```

L'oscillazione di "ritorno" fornirà il valore X=(+1 +1 +1 -1 -1 -1), che, essendo diverso da quello che aveva prima (nell'ultima unità), provocherà un ciclo in più, dopo il quale la rete si rilasserà nell'associazione {X2,Y2}. Come si può notare, la rete ha autocorretto l'errore, riconducendo l'input al dato *piu simile* fra quelli memorizzati. Da questo punto di vista la rete non ha "imparato a memoria" ma ha "ricordato qualcosa di simile", in altre parole, ha *generalizzato* ciò che ha appreso.

Proviamo ora a vedere cosa succede quando dovesse manifestarsi un guasto, ad esempio, l'unità 3 del livello X dà sempre 0 perché non funziona. Partendo dall'input X=(+1 +1 +1 -1 -1 -1) otteniamo: (si noti il terzo addendo):

```
Y[1] = T(+1*2 +1*0  +0 -1*-2 -1*0 -1*-2) = T( 2+0+0+2-0+2)
                                = T(+6) = +1
Y[2] = T(+1*0 +1*-2 +0 -1*0 -1*2 -1*0)   = T( 0-2+0-0-2-0)
                                = T(-4) = -1
Y[3] = T(+1*0 +1*2  +0 -1*0 -1*-2 -1*0)  = T( 0+2+0-0+2-0)
                                = T(+4) = +1
Y[4] = T(+1*-2 +1*0 +0 -1*2 -1*0 -1*2)   = T(-2+0+0-2-0-2)
                                = T(-6) = -1
```

Come è possibile notare, l'output è sorprendentemente corretto. In altre parole il guasto è stato soprasseduto dalla collettività e non ha fatto collassare la rete. Si noti che ogni singola unità non è conscia del risultato globale che la rete sta raggiungendo, ma "fa i fatti suoi". Sembra quasi che vi sia una mano invisibile che guida le unità anarchiche verso una meta comune.

Ma proviamo ora a mettere in crisi questa rete. Cosa succede se le dessimo un input molto diverso da quelli appresi? Proviamo con X=(+1 +1 +1 -1 +1 +1): questo dato è a uguale distanza da X1 e da X2 infatti differisce da entrambi per il 33% circa (due unità diverse).

Andata:

```
Y[1] = T(+1*2 +1*0 +1*2 -1*-2 +1*0 +1*-2) = T( 2+0+2+2+0-2)
                                = T(+6) = +1
Y[2] = T(+1*0 +1*-2 +1*0 -1*0 +1*2 +1*0)  = T( 0-2+0-0+2+0)
                                = T( 0) = -1
Y[3] = T(+1*0 +1*2 +1*0 -1*0 +1*-2 +1*0)  = T( 0+2+0-0-2+0)
                                = T( 0) = -1
Y[4] = T(+1*-2 +1*0 +1*-2 -1*2 +1*0 +1*2) = T(-2+0-2-2+0+2)
                                = T(-6) = -1
```

Ritorno:

```
X[1] = T(+1*2 -1*0 -1*0 -1*-2) = T( 2-0-0+2) = T(+4) = +1
X[2] = T(+1*0 -1*-2 -1*2 -1*0) = T( 0+2-2-0) = T( 0) = -1
X[3] = T(+1*2 -1*0 -1*0 -1*-2) = T( 2-0-0+2) = T(+4) = +1
X[4] = T(+1*-2 -1*0 -1*0 -1*2) = T(-2-0-0-2) = T(-4) = -1
X[5] = T(+1*0 -1*2 -1*-2 -1*0) = T( 0-2+2-0) = T( 0) = -1
X[6] = T(+1*-2 -1*0 -1*0 -1*2) = T(-2-0-0-2) = T(-4) = -1
```

Seconda andata:

```
Y[1] = T(+1*2 -1*0 +1*2 -1*-2 -1*0 -1*-2) = T( 2-0+2+2-0+2)
                                = T(+8) = +1
Y[2] = T(+1*0 -1*-2 +1*0 -1*0 -1*2 -1*0)  = T( 0+2+0-0-2-0)
                                = T( 0) = -1
Y[3] = T(+1*0 -1*2 +1*0 -1*0 -1*-2 -1*0)  = T( 0-2+0-0+2-0)
                                = T( 0) = -1
Y[4] = T(+1*-2 -1*0 +1*-2 -1*2 -1*0 -1*2) = T(-2-0-2-2-0-2)
                                = T(-8) = -1
```

È successo che la rete si è stabilizzata in uno stato *spurio*, cioè non precedentemente memorizzato:

```
X=(+1 -1 +1 -1 -1 -1), Y=(+1 -1 -1 -1)
```

In altre parole, la rete ha fatto confusione ed ha avuto una specie di "allucinazione", nel senso che ha creduto di ricordare qualcosa che non aveva mai visto.

Purtroppo questo non è l'unico inconveniente di questo tipo di rete: problemi analoghi possono affiorare quando si "sovraccarica" una rete nel tentativo di farle apprendere troppa roba. Esistono svariati teoremi a proposito della capacità di memoria di questo tipo di rete [16]: il sovraccarico si manifesta quando il numero di dati da memorizzare si avvicina al numero di unità del livello che ne ha di meno. Teoricamente il limite massimo di dati memorizzabili è:

```
L = min(N,M)
```

dove N e M sono il numero di unità nei livelli X e Y. In sostanza, il numero di unità di una rete limita le sua capacità di memoria. nel nostro caso, avendo N=6 e M=4, si può notare una certa amnesia della rete con 4 dati. Proviamo infatti ad aggiungere alla nostra rete la memorizzazione di queste associazioni:

```
{ X3=(-1 +1 +1 +1 -1 -1), Y3=(-1 +1 +1 -1) }
{ X4=(+1 +1 -1 -1 +1 +1), Y3=(-1 +1 +1 +1) }
```

La matrice risultante sarà:

```
+2  0  0  0
-2  0 +4  0
+2  0  0 -4
-2  0  0  0
 0 +2 -2 +2
-2  0  0 +4
```

Provando a darle l'input X3=(-1 +1 +1 +1 -1 -1) otterremo l'output spurio Y=(-1-1 +1 -1). Anche in questo caso la rete ha avuto le traveggole, nonostante il fatto che l'input fornitole fosse esattamente uno di quelli appresi. Provando ora a farle riapprendere l'associazione {X3,Y3} otteniamo un riacquisto di memoria con conseguente perdita di memoria degli altri dati. In conclusione, la rete manifesta una sorta di *familiarità con gli eventi più frequenti e recenti.*

Poniamoci ora un'altra domanda: supponendo di voler memorizzare nella rete una corretta quantità di associazioni, riuscirà la nostra rete a memorizzare qualsiasi combinazione di dati? Purtroppo no. Se ad esempio provassimo a memorizzare dei dati troppo simili fra di loro, rischiamo di creare confusione. Proviamo ad addestrare la nostra rete con queste tre associazioni (e nessun'altra).

```
{ X1=(+1 -1 +1 -1 +1 -1), Y1=(+1 -1 +1 -1) }
{ X2=(-1 +1 -1 +1 -1 +1), Y1=(-1 +1 -1 +1) }
{ X3=(-1 -1 +1 -1 +1 -1), Y1=(-1 -1 +1 -1) }
```

La matrice dei pesi sarà:

```
+3 -1 +1 -1
-1 +3 -3 +3
+1 -3 +3 -3
-1 +3 -3 +3
+1 -3 +3 -3
-1 +3 -3 +3
```

Nonostante il fatto che le associazioni memorizzate siano appena 3, si possono manifestano errori, infatti, da X=(-1 -1 +1 -1 +1 -1) si ottiene Y=(+1 -1 +1 -1). Questo fatto è dovuto alla estrema somiglianza fra i dati memorizzati, non in termini di ugualianza fra i valori delle unità, ma di varietà nella combinazione dei più e dei meno: in altre parole, i dati X1 e X2 sono simmetrici (cioè hanno la stessa sequenza con segno opposto) e dal punto di vista della rete sono praticamente uguali. La memorizzazione di

un dato implica automaticamente la memorizzazione del suo simmetrico. La rete riesce a memorizzare bene solo i dati con un'alta *ortogonalità* fra di loro. Non bisogna confondere l'ortogonalità di dati con la simmetria: il simmetrico di un dato si ottiene scambiando tutti i +1 con -1 (e viceversa); l'ortogonale di un dato si ottiene scambiandone circa la metà.

Un'utile metafora per capire il comportamento di questa rete neurale è quella di immaginare il supporto di memoria come un foglio di gomma in cui vi siano alcune pietre che creano dei crateri. Ogni cratere rappresenta un'associazione memorizzata; l'esecuzione della rete può essere immaginata come il movimento di una pallina su questa superficie: se la pallina è fin dall'inizio dentro un cratere, vi rimane, e questo corrisponde a far eseguire la rete con un input che coincide esattamente con uno di quelli appresi. Se la pallina è nelle vicinanze di alcuni crateri (input lievemente distorto), rotola dentro al cratere più vicino (cioè la rete fa più oscillazioni). Se è distante da tutti i crateri (input molto distorto), la pallina vaga un po' e poi si assesta in qualche avvallamento spurio. Se due crateri sono troppo vicini (dati non ortogonali), oppure se vi sono troppi crateri, si creano strane colline con avvallamenti spuri. Il fatto che prima o poi la pallina si fermi è garantito dal fatto che il foglio di gomma non ha buchi né discese infinite, ma solo crateri.

Questa metafora non è immaginaria, ma frutto di una singolare analogia fra lo stato della rete e il concetto di energia di un sistema dinamico. È possibile definire una sorta di *energia della rete* come una grandezza che tiene conto del valore di ogni unità e delle relative connessioni:

$$E = -\sum_i \sum_j W_{ij} X_i Y_j$$

Il valore di E in ogni istante dell'esecuzione della rete rappresenta la posizione della pallina sul foglio di gomma. A questo punto è piuttosto facile dimostrare che la nostra rete ha una "superficie d'energia" con crateri e senza buchi (questo è il teorema della convergenza, cioè quello che dimostra che le oscillazioni della rete prima o poi terminano). La dimostrazione si basa sul fatto che un qualsiasi cambiamento nella rete comporta una diminuzione dell'energia e che esiste un minimo (il cratere): di conseguenza, comunque vadano le cose, la rete ten-

derà a raggiungere il suo minimo energetico e a rimanervi. Consideriamo un qualsiasi cambiamento che avviene nella rete: sarà senz'altro dovuto al fatto che una certa unità, diciamo X_i, riceve uno stimolo (dalle altre unità) non concorde con il suo attuale valore, cioè di diverso segno. Ad esempio, se l'unità X_i vale +1 e riceve uno stimolo $\sum_j W_{ij}Y_j = -4$, verrà trasformata in -1. Calcoliamo adesso la variazione dell'energia in seguito a questa modifica: con semplici passaggi si può calcolare che la variazione ΔE dell'energia dovuta alla variazione ΔX_i dell'i-esima unità del livello X è:

$$\Delta E = -\Delta X_i \sum_j W_{ij}Y_j$$

Dall'osservazione fatta prima, si può notare che ΔX_i ha sempre lo stesso segno di $\sum_j W_{ij}Y_j$, quindi la loro moltiplicazione è sempre positiva (vi ricordate: più per più fa più, meno per meno fa più?). Di conseguenza ΔE è sempre negativa o al limite nulla. Abbiamo dimostrato che un qualsiasi cambiamento dovuto all'esecuzione della rete tende a far diminuire l'energia della rete fino ad un minimo stabile, quindi è garantito che la rete non oscilla casualmente ma tende sempre a "rilassarsi" verso uno stato stabile.

Ritornando agli aspetti generali, possiamo concludere che con le reti neurali si può realizzare delle memorie associative con caratteristiche molto simili alle memorie degli esseri viventi, vale a dire poco precise, poco capienti, ma molto elastiche, robuste e adattabili.

5 Simulazione software di una memoria associativa

In questo capitolo descriviamo due programmi scritti in linguaggio Pascal per la simulazione sequenziale su computer di una memoria associativa neurale. La relativa versione eseguibile per Personal Computer MS-DOS è disponibile al pubblico attraverso le strutture divulgative di RAI-TELEVIDEO. Il primo di questi due programmi si chiama MEM_ASS e permette di sperimentare una memoria associativa fornendo "a mano" i valori +1 e -1 dei livelli X e Y. Il listato 1 contiene il codice Pascal di questo programma.

Listato 1:

```
(*-----------------------------------------------------------------
                    ESERCIZIO SU RETI NEURALI
                 MEMORIA ASSOCIATIVA DISTRIBUITA
                       PROGRAMMA MEM_ASS
                       A. Mazzetti - 1991
-----------------------------------------------------------------*)
program mem_ass;
var X:array[1..100]of integer;          (* livello X *)
   Y:array[1..100]of integer;           (* livello Y *)
   W:array[1..100,1..100]of integer;    (* matrice dei pesi *)
   N,M:integer;                         (* numero di unità *)
   com:char;                            (* comando utente *)
   i,j:integer;                         (* indici *)
```

```
(*------------------------- acquisisci_X -------------------------*)
procedure acquisisci_X;
 begin
   for i:=1 to N do begin
     write('X',i,' ? : '); readln(X[i])
   end;
   writeln;
 end;


(*------------------------- acquisisci_Y -------------------------*)
procedure acquisisci_Y;
 begin
   for j:=1 to M do begin
     write('Y',j,' ? : '); readln(Y[j])
   end;
   writeln
 end;


(*------------------------- apprendi ------------------------------*)
procedure apprendi;
 begin
   acquisisci_X;
   acquisisci_Y;
   for j:=1 to M do
     for i:=1 to N do
       W[i,j]:=W[i,j]+(X[i]*Y[j]);
 end;


(*----------------- funzione di trasferimento T -----------------*)
function T(A:integer):integer;
 begin
   if A>0 then T:= +1 else T:= -1
 end;


(*------------------------- emetti_output -------------------------*)
procedure emetti_output;
 begin
     for j:=1 to M do writeln(Y[j]:6);
     writeln;
 end;
```

```
(*------------------------- esegui -------------------------*)
procedure esegui;
 var A:integer; cambiato:boolean;
 begin
   acquisisci_X;
   for j:=1 to M do Y[j]:=0;
   repeat
     cambiato:=false;
     for j:=1 to M do begin
       A:=0;
       for i:=1 to N do A:=A+(W[i,j]*X[i]);
       Y[j]:=T(A);
     end;
     for i:=1 to N do begin
       A:=0;
       for j:=1 to M do A:=A+(W[i,j]*Y[j]);
       if T(A)<>X[i] then cambiato:=true;
       X[i]:=T(A);
     end;
     emetti_output;
   until not cambiato;
 end;

(*------------------------- modifica -------------------------*)
procedure modifica;
 begin
   write('Numero di unità del livello X ? (1..100): ');readln(N);
   write('Numero di unità del livello Y ? (1..100): ');readln(M);
   for i:=1 to N do X[i]:=0;
   for j:=1 to M do Y[j]:=0;
   for i:=1 to N do
     for j:=1 to M do
       W[i,j]:=0;
 end;

(*------------------------- visualizza -------------------------*)
procedure visualizza;
 begin
   writeln; writeln('unità di input : ');
   for i:=1 to N do write(X[i]:2,' '); writeln;
   writeln('unità di output: ');
   for j:=1 to M do write(Y[j]:2,' '); writeln;
   writeln; write(' ':4);
   for j:=1 to M do write(j:2,'Y '); writeln;writeln;
   for i:=1 to N do begin
```

```
      write(i:2,'X ');
      for j:=1 to M do write(W[i,j]:3,' '); writeln;
    end;
    writeln;
  end;

(*------------------------- salva -------------------------------*)
procedure salva;
  var f:text; fn:string[20];
  begin
    write('Nome del file in cui salvare i pesi : ');readln(fn);
    assign(f,fn); rewrite(f);
    writeln(f,N);writeln(f,M);
    for j:=1 to M do
      for i:=1 to N do
        writeln(f,W[i,j]:6);
    close(f);
  end;

(*------------------------- carica ------------------------------*)
procedure carica;
  var f:text; fn:string[20]; r:integer;
  begin
    write('Nome del file da cui caricare i pesi : ');readln(fn);
    assign(f,fn); reset(f);
    readln(f,r);
    if r=N then begin
      readln(f,r);
      if r=M then begin
        for j:=1 to M do
          for i:=1 to N do
            readln(f,W[i,j]);
      end else writeln('errore: M letto da file è ,r,' anziché ',M)
    end else writeln('errore: N letto da file è ,r,' anziché ',N);
    close(f);
  end;

(*------------------------- main --------------------------------*)
begin
  modifica;
  repeat
    writeln;write('Apprendi Esegui Visualizza Salva Carica
                                    Modifica Fine: ');
    readln(com);
    case com of
```

```
    'a':apprendi;
    'e':esegui;
    'v':visualizza;
    's':salva;
    'c':carica;
    'm':modifica;
    end;
  until com='f'
end.
```

Listato 1: *Codice Pascal del programma MEM_ASS per la simulazione sequenziale di memorie associative neurali.*

Appena lanciato, questo programma chiede all'utente il numero di unità dei livelli X e Y e poi appare un menù:

```
C:> MEM_ASS
Numero di unità del livello X ? (1..100): 6
Numero di unità del livello Y ? (1..100): 4
Apprendi Esegui Visualizza Salva Carica Modifica Fine: _
```

Qui bisogna rispondere con una lettera minuscola (a,e,v,s,c,m,f): il comando "a" (Apprendi) acquisisce il valore delle unità dei livelli X e Y e poi fa apprendere dalla rete l'associazione, ad esempio:

```
Apprendi Esegui Visualizza Salva Carica Modifica Fine: a
X1 ? : 1
X2 ? : -1
X3 ? : 1
X4 ? : -1
X5 ? : 1
X6 ? : -1
Y1 ? : 1
Y2 ? : 1
Y3 ? : -1
Y4 ? : -1
Apprendi Esegui Visualizza Salva Carica Modifica Fine: v
unità di input:
 1 -1  1 -1  1 -1
unità di output:
 1  1 -1 -1
```

```
     1Y  2Y  3Y  4Y
 1X   1   1  -1  -1
 2X  -1  -1   1   1
 3X   1   1  -1  -1
 4X  -1  -1   1   1
 5X   1   1  -1  -1
 6X  -1  -1   1   1
Apprendi Esegui Visualizza Salva Carica Modifica Fine: _
```

Si noti l'uso del comando "v" (Visualizza) per avere una stampa a video della matrice dei pesi. Il comando "e" (Esegui) acquisisce il valore delle unità del livello X e poi fa partire la rete: ad ogni oscillazione viene visualizzato il livello Y. Nel listato 1 si può notare nella procedura "apprendi" la formula di Hebb e nella procedura "esegui" il loop in cui vengono fatte le "andate" e i "ritorni" fino a quando non cambia più nulla. Si noti anche la funzione di trasferimento T.

I due comandi "s" e "c" (Salva e Carica) servono per salvare su file e caricare da file la matrice dei pesi W e il numero delle unità N e M. Il comando "m" (Modifica) permette di ridefinire il numero delle unità dei due livelli e riazzera la matrice; il comando "f" (Fine) fa terminare il programma.

L'altro programma si chiama IMMAGINI ed è una applicazione della memoria associativa nel campo del riconoscimento delle immagini (figura 13).

Qui i dati da associare sono matrici di pixel in cui ad ogni pixel corrisponde una unità delle rete: al colore bianco corrisponde il valore +1 e al nero -1. Un esempio di possibile associazione fra due immagini può essere quella fra il minuscolo e il maiuscolo di caratteri:

```
...XXX..X...      ............
..X...X.X...      .....XX.....
.X.....XX...      ....X..X....
X......XX...      ...X....X...
X......XX...      ..XXXXXXXX..
.X.....XX...      .X........X.
..X...X.X..X      X..........X
...XXX...XX.      ............
```

Ogni immagine deve risiedere su un file di testo in cui i primi due numeri devono contenere il numero di righe e di colonne e le successive righe contengono l'immagine espressa attraverso i

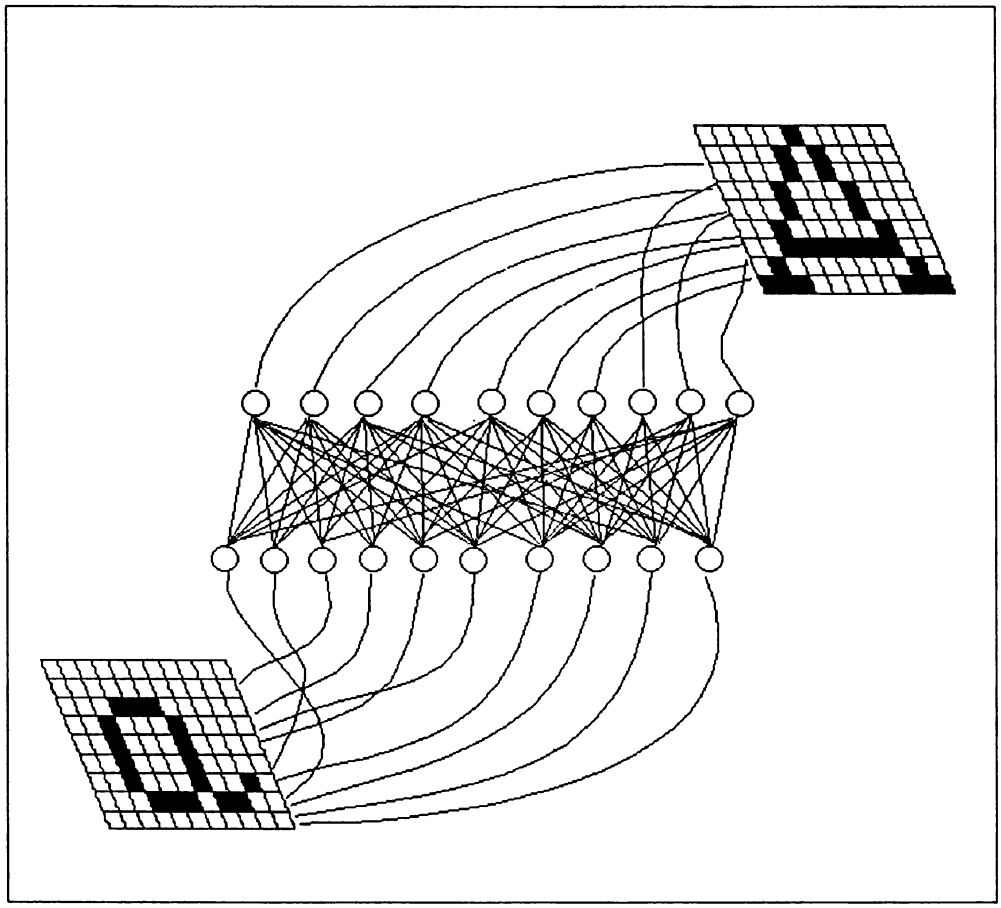

Figura 13. *Un'applicazione della memoria associativa nel campo del riconoscimento delle immagini.*

caratteri "punto" e "X maiuscola"; tale file deve essere realizzato dall'utente tramite un qualsiasi editor. Attenzione a non confondere questa matrice di pixel con la matrice dei pesi della rete.

Lanciando il programma IMMAGINI si ottiene un menù uguale a quello del programma MEM_ASS, però cambia il tipo di elaborazione (vedi listato 2): il comando "a" (Apprendi) infatti non chiede più l'inserimento manuale dei valori +1 e -1, ma i nomi dei file che contengono le immagini. Il comando "e" (Esegui) richiede il nome del file dell'immagine X e poi sputa fuori a video l'immagine Y associata. Gli altri comandi sono uguali a quelli già visti nel programma MEM_ASS.

Listato 2:

```
(*-------------------------------------------------------------------
                    ESERCIZIO SU RETI NEURALI
                 MEMORIA ASSOCIATIVA DISTRIBUITA
                      PROGRAMMA IMMAGINI
                      A. Mazzetti - 1991
-------------------------------------------------------------------*)

program immagini;
var X:array[1..100]of integer;              (* livello X *)
   Y:array[1..100]of integer;               (* livello Y *)
   W:array[1..100,1..100]of integer;        (* matrice dei pesi *)
   N,M:integer;                             (* numero di unità *)
   com:char;                        (* comandi e modalità di I/O *)
   i,j,rig_x,col_x,rig_y,col_y,r,c:integer;     (* indici vari *)

(*---------------------- acquisisci_X ----------------------*)
procedure acquisisci_X;
 var k:char; f:text; fn:string[20]; nn:integer;
 begin
   write('Nome file dell''immagine X : ');readln(fn);
   assign(f,fn); reset(f);
   readln(f,nn);
   if nn=col_x then begin
     readln(f,nn);
     if nn=rig_x then begin
       i:=1;
       for r:=1 to rig_x do begin
         for c:=1 to col_x do begin
           read(f,k);write(k);
           if k='.' then X[i]:=-1 else X[i]:=1;
           i:=i+1;
         end;
         readln(f);writeln;
       end
     end else writeln ('errore: il file ha ',nn,' righe')
   end else writeln('errore: il file ha ',nn,' colonne');
   close(f);
 end;

(*---------------------- acquisisci_Y ----------------------*)
procedure acquisisci_Y;
 var k:char; f:text; fn:string[20]; nn:integer;
```

```
begin
  write('nome file dell''immagine Y : ');readln(fn);
  assign(f,fn); reset(f);
  readln(f,nn);
  if nn=col_y then begin
    readln(f,nn);
    if nn=rig_y then begin
      j:=1;
      for r:=1 to rig_y do begin
        for c:=1 to col_y do begin
          read(f,k);write(k);
          if k='.' then Y[j]:=-1
          else if k='?' then Y[j]:=0
          else Y[j]:=1;
          j:=j+1;
        end;
        readln(f);writeln;
      end
    end else writeln ('errore: il file ha ',nn,' righe')
  end else writeln('errore: il file ha ',nn,' colonne');
  close(f);
end;


(*------------------------- apprendi ------------------------------*)
procedure apprendi;
 begin
   acquisisci_X;
   acquisisci_Y;
   for j:=1 to M do
     for i:=1 to N do
       W[i,j]:=W[i,j]+(X[i]*Y[j]);
 end;


(*----------------- funzione di trasferimento T -----------------*)
function T(A:integer):integer;
 begin
   if A>0 then T:= +1 else T:= -1
 end;
```

```
(*------------------------- emetti_output -------------------------*)
procedure emetti_output;
 begin
   j:=1; writeln;
   for r:=1 to rig_y do begin
     for c:=1 to col_y do begin
       if Y[j]>0 then write('X') else write('.');
       j:=j+1;
     end;
     writeln;
   end;
 end;


(*------------------------- esegui -------------------------*)
procedure esegui;
 var A:integer; cambiato:boolean;
 begin
   acquisisci_X;
   for j:=1 to M do Y[j]:=0;
   repeat
     cambiato:=false;
     for j:=1 to M do begin
       A:=0;
       for i:=1 to N do A:=A+(W[i,j]*X[i]);
       Y[j]:=T(A);
     end;
     for i:=1 to N do begin
       A:=0;
       for j:=1 to M do A:=A+(W[i,j]*Y[j]);
       if T(A)<>X[i] then cambiato:=true;
       X[i]:=T(A);
     end;
     emetti_output;
   until not cambiato;
 end;


(*------------------------- modifica -------------------------*)
procedure modifica;
 begin
   write('Quante righe nell''immagine X ? : ');   readln(rig_x);
   write('Quante colonne nell''immagine X ? : '); readln(col_x);
   writeln;
```

```
  write('Quante righe nell''immagine Y ? : ');   readln(rig_y);
  write('Quante colonne nell''immagine Y ? : '); readln(col_y);
  N:=rig_x*col_x;
  M:=rig_y*col_y;
  if (N>100)or(M>100) then writeln('Immagine troppo grossa.
                                   Ripetere.');
  if (N<100)and(M<199) then begin
    for i:=1 to N do X[i]:=0;
    for j:=1 to M do Y[j]:=0;
    for i:=1 to N do
      for j:=1 to M do
        W[i,j]:=0;
  end;
end;


(*------------------------- visualizza -------------------------*)
procedure visualizza;
 begin
   writeln; writeln('unità di input : ');
   for i:=1 to N do write(X[i]:2,' '); writeln;
   writeln('unità di output: ');
   for j:=1 to M do write(Y[j]:2,' '); writeln;
   writeln; write(' ':4);
   for j:=1 to M do write(j:2,'Y '); writeln;writeln;
   for i:=1 to N do begin
     write(i:2,'X ');
     for j:=1 to M do write(W[i,j]:3,' '); writeln;
   end;
   writeln;
 end;


(*------------------------- salva -------------------------*)
procedure salva;
 var f:text; fn:string[20];
 begin
   write('Nome del file in cui salvare i pesi : ');readln(fn);
   assign(f,fn); rewrite(f);
   writeln(f,N);writeln(f,M);
   for j:=1 to M do
     for i:=1 to N do
       writeln(f,W[i,j]:6);
   close(f);
 end;
```

```
(*------------------------- carica ------------------------------*)
procedure carica;
 var f:text; fn:string[20]; r:integer;
 begin
   write('Nome del file da cui caricare i pesi : ');readln(fn);
   assign(f,fn); reset(f);
   readln(f,r);
   if r=N then begin
     readln(f,r);
     if r=M then begin
       for j:=1 to M do
         for i:=1 to N do
           readln(f,W[i,j]);
     end else writeln('errore: M letto da file _ ',r,' anzich_
                                        ',M)
   end else writeln('errore: N letto da file è ',r,' anziché ',N);
   close(f);
 end;


(*------------------------- main ------------------------------*)
begin
 modifica;
 repeat
   writeln;write('Apprendi Esegui Visualizza Salva Carica
                                        Modifica Fine: ');
   readln(com);
   case com of
   'a':apprendi;
   'e':esegui;
   'v':visualizza;
   's':salva;
   'c':carica;
   'm':modifica;
   end;
 until com='f'
end.
```

Listato 2: *Codice Pascal del programma IMMAGINI per la associazione di immagini.*

Supponiamo di avere creato i seguenti file di immagini:

```
file A.X e A.Y:
8                   8
12                  12
...XXX..X...        ............
..X...X.X...        .....XX.....
.X.....XX...        ....X..X....
X......XX...        ...X....X...
X......XX...        ..XXXXXXXX..
.X.....XX...        .X........X.
..X...X.X..X        X..........X
...XXX...XX.        ............

file B.X e B.Y:

8                   8
12                  12
.....XX.....        .XXXXXX.....
....X..X....        .X.....X....
...X...X....        .X.....X....
..X...XX....        .XXXXXX.....
..X.XX......        .X.....X....
..XX...XXX..        .X......X...
XX.X...X....        .X.....X....
....XXX.....        .XXXXXX.....

file C.X e C.Y:

8                   8
12                  12
............        ..XXXXXXX...
....XXXX....        .X.......X..
..XX....X...        X...........
.X..........        X...........
X...........        X...........
.X..........        X...........
..XX....X...        .X.......X..
....XXXX....        ..XXXXXXX...
```

Supponiamo di avere alcune versioni distorte di queste immagini:

```
file A1.X:

...XXX..X...
..X...X.X...
......XX....
X......XX...
X....X.XXX..
.X..X..XX...
..X...X.X...
...XXX....X.

file B1.X:

.....XX.....
....XX.X....
XXXXXXXXXXXX
..X..XXX....
..X.XX......
..XX.X.XXX..
XX.X.X.X....
....XXX.....
```

Quella che si trova di seguito è una traccia dell'uso del programma IMMAGINI:

```
C:> IMMAGINI
Quante righe nell'immagine X ? : 8
Quante colonne nell'immagine X ? : 12
Quante righe nell'immagine Y ? : 8
Quante colonne nell'immagine Y ? : 12

Apprendi Esegui Visualizza Salva Carica Modifica Fine: a
Nome file dell'immagine X : A.X

...XXX..X...
..X...X.X...
.X.....XX...
X......XX...
X......XX...
.X.....XX...
..X...X.X..X
...XXX...XX.
```

```
Nome file dell'immagine Y : A.Y

............
.....XX.....
....X..X....
...X....X...
..XXXXXXXX..
.X........X.
X..........X
............
Apprendi Esegui Visualizza Salva Carica Modifica Fine: a
Nome file dell'immagine X : B.X

.....XX.....
....X..X....
...X...X....
..X...XX....
..X.XX......
..XX...XXX..
XX.X...X....
....XXX.....

nome file dell'immagine Y : B.Y
.XXXXXX.....
.X.....X....
.X.....X....
.XXXXXX.....
.X.....X....
.X......X...
.X.....X....
.XXXXXX.....

Apprendi Esegui Visualizza Salva Carica Modifica Fine: a
Nome file dell'immagine X : C.X

............
....XXXX....
..XX....X...
.X..........
X...........
.X..........
..XX....X...
....XXXX....
```

```
nome file dell'immagine Y : C.Y

..XXXXXXX...
.X.......X..
X...........
X...........
X...........
X...........
.X.......X..
..XXXXXXX...

Apprendi Esegui Visualizza Salva Carica Modifica Fine: e
Nome file dell'immagine X : A.X

...XXX..X...
..X...X.X...
.X.....XX...
X......XX...
X......XX...
.X.....XX...
..X...X.X..X
...XXX...XX.

............
.....XX.....
....X..X....
...X....X...
..XXXXXXXX..
.X........X.
X..........X
............

Apprendi Esegui Visualizza Salva Carica Modifica Fine: e
Nome file dell'immagine X : B.X
.....XX.....
....X..X....
...X...X....
..X...XX....
..X.XX......
..XX...XXX..
XX.X...X....
....XXX.....
```

```
.XXXXXX.....
.X.....X....
.X.....X....
.XXXXXX.....
.X.....X....
.X......X...
.X.....X....
.XXXXXX.....
```

Apprendi Esegui Visualizza Salva Carica Modifica Fine: e
Nome file dell'immagine X : A1.X

```
...XXX..X...
..X...X.X...
......XX....
X......XX...
X....X.XXX..
.X..X..XX...
..X...X.X...
...XXX....X.

............
.....XX.....
....X..X....
...X....X...
..XXXXXXXX..
.X........X.
X..........X
............

............
.....XX.....
....X..X....
...X....X...
..XXXXXXXX..
.X........X.
X..........X
............
```

Apprendi Esegui Visualizza Salva Carica Modifica Fine: e
Nome file dell'immagine X : B1.X

```
.....XX.....
....XX.X....
XXXXXXXXXXXX
..X..XXX....
..X.XX......
..XX.X.XXX..
XX.X.X.X....
....XXX.....

.XXXXXX.....
.X.....X....
.X.....X....
.XXXXXX.....
.X.....X....
.X......X...
.X.....X....
.XXXXXX.....

.XXXXXX.....
.X.....X....
.X.....X....
.XXXXXX.....
.X.....X....
.X......X...
.X.....X....
.XXXXXX.....

Apprendi Esegui Visualizza Salva Carica Modifica Fine: s
Nome del file in cui salvare i pesi : immagini.w
Apprendi Esegui Visualizza Salva Carica Modifica Fine: f
C:>
```

6 Reti "back propagation"

Il modello di rete neurale visto nel capitolo precedente è molto semplice è facile da capire, ma anche molto limitato. La semplicità risiede sia nella topologia di rete (due soli livelli con connessioni totali bidirezionali), sia nelle singole unità (solo valori +1 e -1, con funzione di trasferimento a gradino), sia nella legge di aprendimento (formula di Hebb). I limiti di questa rete sono molto pesanti: non riesce ad apprendere tutto ma solo dati ortogonali, può eseguire solo associazioni ma non funzioni qualsiasi, è lenta in esecuzione perché oscilla.

L'obiettivo che ci prefiggiamo ora è lo studio di un modello di rete più generale e più affidabile: ad esempio una rete che riconosce un'immagine dovrà avere nel livello X tante unità quanti sono i pixel e nel livello Y una sola unità che indica un si/no. Essendoci una sola unità nel livello Y, vengono a cadere tutte le garanzie del modello di memoria associativa: è quindi auspicabile un modello di rete più generale.

Rispolveriamo qualche teorema visto nel capitolo 3 e ragioniamoci su. Innanzitutto consideriamo la topologia di rete: alcuni teoremi affermano che una rete con due soli livelli non potrà mai apprendere certe funzioni, chiamate dai matematici funzioni "linearmente non separabili". Un facile esempio di funzione linearmente non separabile è la funzione logica XOR (in italiano "o esclusivo", in latino "aut aut"); l'espressione A XOR B vale vero quando è vero A o B ma non tutti e due. Senza addentrarci ulteriormente nel merito di questa classe di funzioni, osserviamo che l'architettura a livelli di una rete limita le sue capacità di calcolo. Fortunatamente, però, l'adozione di più di due livelli permette di superare queste difficoltà.

Come già visto nel capitolo 3, il teorema di Hecht-Nielsen afferma che una qualsiasi funzione F può essere computata da una rete neurale organizzata in tre livelli, con connessioni totali fra i livelli senza cicli (rete non ricorrente, "feed forward network"). Con un'architettura di questo tipo vengono eliminati sia i problemi di ortogonalità dei dati da apprendere, sia i problemi di capacità di memoria della rete, infatti, l'inserimento di *unità nascoste* nel livello intermedio fornisce alla rete la potenza di calcolo necessaria per svolgere qualsiasi funzione. In altri termini, il livello intermedio di unità serve alla rete per formarsi una sua rappresentazione interna del problema, necessaria per "capire cosa sta facendo" e fornire un output appropriato. In alcuni casi può essere conveniente adottare più livelli intermedi anziché un solo livello di grosse dimensioni: per semplicità non tratteremo questi casi in questa sede.

Un'altra miglioria che si può introdurre in una rete neurale riguarda il valore delle singole unità: anziché valori bipolari {-1,+1}, adottiamo valori reali qualsiasi compresi in un certo intervallo, ad esempio {0..1}. In questo modo, l'input X e l'output Y della rete sono insiemi di numeri reali anziché di valori binari, con notevole vantaggio di espressività. L'introduzione di valori reali implica la necessità di modificare anche la forma della funzionedi trasferimento T delle unità: non più a gradino, ma a *sigmoide*. Una funzione di trasferimento sperimentata con successo è la seguente:

$$T = \frac{1}{1+e^{-A}}$$

In figura 14 compare il grafico di questa funzione.

Anche questa funzione, come quella a gradino delle memorie associative, è "centrata" sullo zero; questo implica che ogni unità della rete è "accesa" (più o meno intensamente) quando la sua attivazione è positiva ed è "spenta" quando quando negativa. Questo non è sempre l'ideale, perché in molte applicazioni sarebbe più opportuno avere delle soglie su valori qualsiasi diversi per ogni unità. Ad esempio, la k-esima unità tende ad accendersi quando la sua attivazione è vicina al valore –18; vale a dire, una sigmoide centrata sul valore A_k=–18. Per tenere conto di questo fatto modifichiamo la formula dell'attivazione delle unità in questo modo:

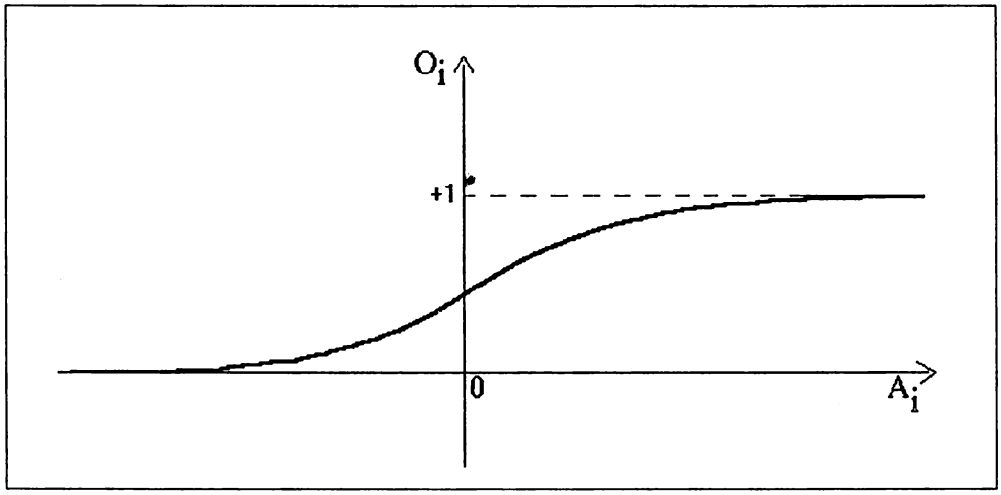

Figura 14 *Funzione di tasferimento a sigmoide per unità con valore di attivazione reale fra zero e uno.*

$$A_k = \sum_i W_{ik} X_i - B_k$$

dove il termine B_k rappresenta la soglia della k-esima unità, cioè il punto dove centrare la sigmoide; chiameremo tale soglia *Bias*, dall'inglese "tendenza", per esprimere il fatto che in assenza di stimoli l'unità tende ad assumere quel valore.

I bias delle diverse unità possono essere diversi fra loro e dipendono dall'applicazione, cioè dalla funzione che la rete deve saper svolgere. In altri termini, il valore B_k dovrà essere appreso alla stessa stregua dei pesi W: questo suggerisce di considerare il bias come il peso di una connessione fittizia verso una "unità fantasma" che vale sempre 1. Supponendo di avere N unità nel livello X di una rete, il bias sarà realizzato attraverso una (N+1)-esima unità il cui valore è fisso ad 1 e le cui connessioni verso le altre unità rappresentano l'opposto del loro bias. La formula dunque ritorna ad essere quella di prima:

$$A_k = \sum_{i=1}^{N+1} W_{ik} X_i$$

con l'unica differenza che l'indice i varia da 1 a N+1 anziché N.

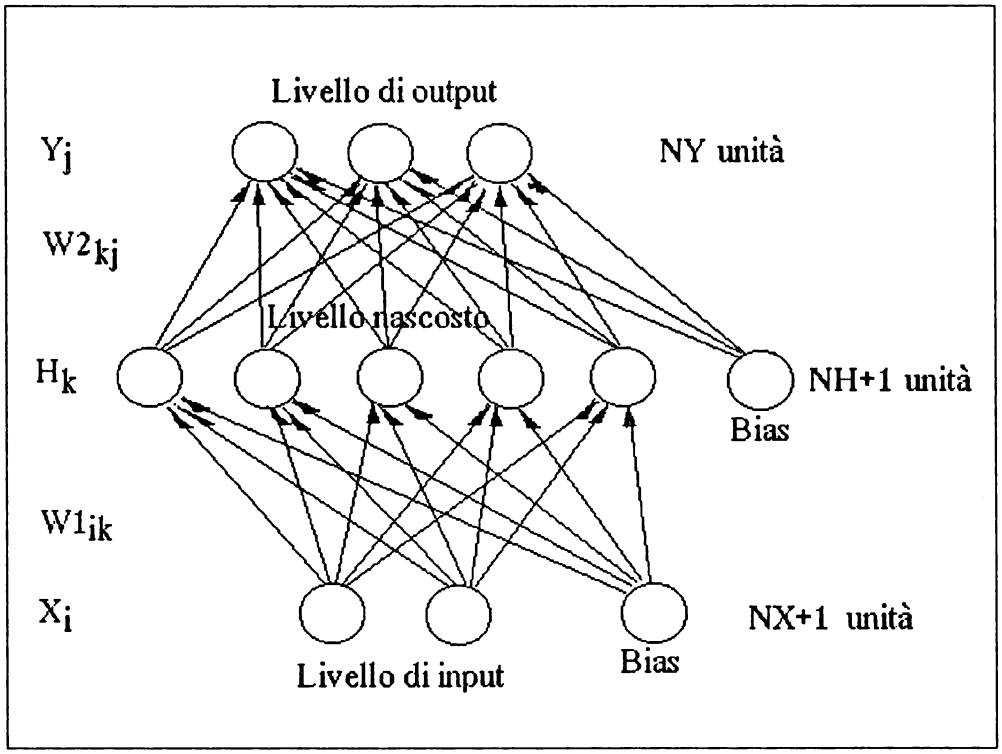

Figura 15. *Struttura di una rete back propagation*

Riassumendo, per ottenere una rete in grado di eseguire una qualsiasi funzione Y=F(X) è necessaria una struttura come quella di in figura 15, in cui le unità hanno valori di attivazione reali, la funzione di trasferimento è a sigmoide e la topologia è organizzata su tre livelli di cui uno di input (livello X, avente NX unità più il bias), uno intermedio (livello H, dall'inglese Hidden=nascosto, avente NH unità più il bias) e uno di output (livello Y, con NY unità). La relativa struttura dati per computer sarà:

```
X:array[1..NX+1] of real; (* unità del livello X + bias*)
H:array[1..NH+1] of real; (* unità del livello H + bias*)
Y:array[1..NY]   of real; (* unità del livello Y*)
W1:array[1..NX+1,1..NH] of real;
                          (*matrice dei pesi primo livello*)
W2:array[1..NH+1,1..NY] of real;
                          (*matrice dei pesi secondo
                            livello*)
```

A differenza del modello di rete visto per le memorie associative, questo modello è privo di connessioni cicliche, infatti i legami fra le unità sono solo in una direzione; questo implica che l'esecuzione della rete sarà estremamente veloce, infatti avviene in una sola passata (per essere precisi, essendoci tre livelli di unità, vi saranno due passaggi: uno dal livello di input a quello nascosto e un altro al livello di output). L'algoritmo di esecuzione della rete sarà dunque:

```
procedura esegui;
  acquisisci dato X;
  azzera livelli H e Y;
  per ogni unità k del livello H
    calcola la sua attivazione (con bias)
                        A=SUMi(W1[i,k]*X[i]);
    H[k]=T(A);
  fine;
  per ogni unità j del livello Y
    calcola la sua attivazione (con bias)
                        A=SUMk(W2[k,j]*H[k]);
    Y[j]=T(A);
  fine;
  visualizza dato Y;
fine;

funzione T(A);
  T=1/(1+exp(-A));
fine;
```

Ciò che è ciclico nelle reti back propagation è il meccanismo di apprendimento: infatti, per ottenere una giusta combinazione dei pesi non è più sufficiente la formula di Hebb, ma è necessaria una formula che operi per *epoche* (vedi capitolo 3), vale a dire che aggiusti i pesi per piccoli passi ciclando più volte sugli esempi da apprendere.

Il meccanismo di apprendimento funziona in questo modo: dato un insieme di esempi nella forma {X,D}, ove X rappresenta l'input alla rete e D l'output desiderato, la rete inizialmente prova ad eseguire l'input X e vede quale output Y salta fuori con gli attuali pesi; questo viene confrontato con l'output desiderato D e si misura l'errore commesso; quest'ultimo viene "propagato all'indietro" aggiustando i pesi (da qui il nome "back propagation"). Il tutto viene ripetuto per ogni esempio e costituisce una singola epoca di apprendimento. Le epoche

vengono poi ripetute fino a che l'errore commesso non rimane sotto ad un certo errore ammesso. L'algoritmo di simulazione di questo processo è:

```
procedure apprendi;
  ripeti (* epoche *)
    per ogni esempio {X,D}
      esegui la rete con X e trova Y;
      calcola errore della rete nel singolo esempio;
      aggiusta i pesi W1 e W2;
    fine;
    calcola errore dell'intera epoca;
  finché errore dell'epoca inferiore ad errore ammesso
fine;
```

Vediamo ora come effettuare la modifica dei pesi. Ricordando che le unità hanno valori continui fra zero e uno, possiamo definire l'errore E_j commesso dalla j-esima unità del livello Y come differenza dal valore desiderato, cioè:

$$E_j = D_j - Y_j$$

L'errore sarà dunque un numero compreso fra -1 e +1, in cui zero significa che non si è avuto alcun errore, -1 e +1 significano che si ha sbagliato del tutto in un senso o nell'altro. L'errore globale dell'intera rete durante la presentazione di un singolo esempio è concepibile come la somma degli errori E_j commessi da ogni unità Y_j, però se facessimo una semplice somma algebrica, rischieremmo di annullare gli errori opposti: ad esempio, se Y_1 ha commesso un errore di +1 e Y_2 un errore di -1, la loro somma è zero e questo potrebbe essere interpretato come assenza d'errore. Per ovviare a questo inconveniente, è sufficiente definire l'errore globale della rete E come somma dei quadrati degli errori, ottenendo una grandezza più significativa (per semplificare alcuni calcoli è più opportuno dividerlo per due):

$$E = \frac{1}{2}\sum_j \left(D_j - Y_j\right)^2$$

Supponiamo, quindi, che in seguito alla presentazione di un certo esempio la rete abbia commesso un certo errore E: questo è dovuto al fatto che i pesi delle connessioni non erano buoni. Consideriamo uno qualsiasi di questi pesi, ad esempio quello che connette H_k con Y_j, cioè $W2_{kj}$. Se $W2_{kj}$ fosse stato diverso, la rete avrebbe dato un errore diverso. Consideriamo l'andamento

dell'errore E al variare del valore di $W2_{kj}$: questo avrà una certa forma strana come quella di figura 16.

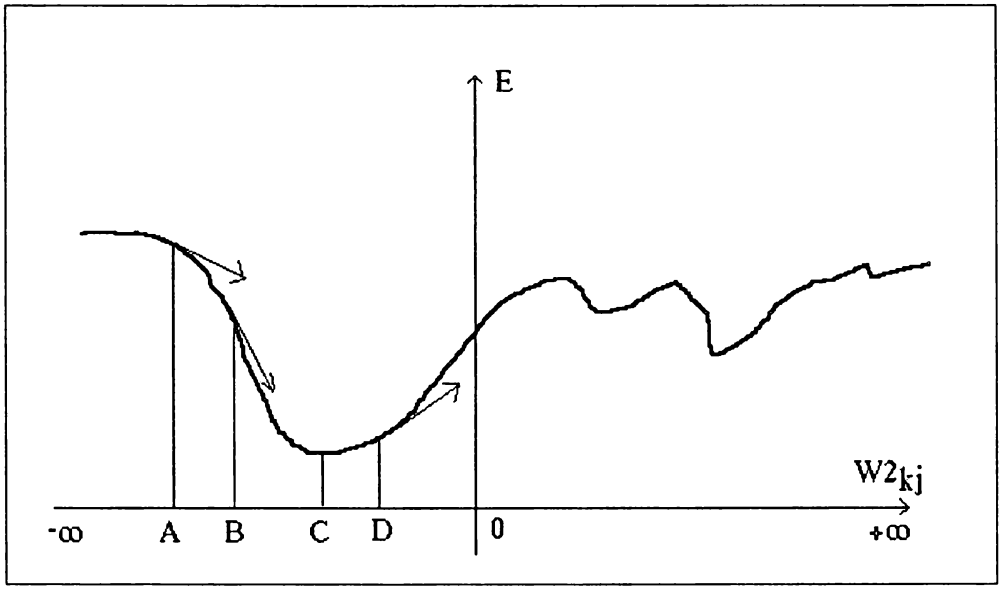

Figura 16. *Andamento dell'errore della rete in funzione di un peso.*

Il valore di $W2_{kj}$ corrispondente al minimo di quella curva (punto C in figura 16) è proprio quello che cerchiamo: se tutti i pesi della rete fossero quelli corrispondenti al rispettivo minimo, la rete non darebbe più errore, cioè avrebbe imparato correttamente. Il problema è dunque quello di trovare il valore di $W2_{kj}$ corrispondente al minimo; si tenga presente però che la forma della funzione dipende da tutti gli altri pesi della rete e siccome da un'epoca all'altra ogni peso della rete cambia, in ogni epoca avrò a che fare con funzioni di forma sempre diversa. Il vero problema è quello di trovare il minimo di una funzione in uno spazio N-dimesionale, che si traduce nel problema di trovare un insieme di minimi che vadano d'accordo fra di loro in spazi bidimensionali. Purtroppo non esiste una formula magica per trovare quest'insieme di valori, ma fortunatamente esiste una tecnica per arrivarci per piccoli passi, tecnica che i matematici chiamano *discesa del gradiente.*

L'idea di base di questa tecnica è quella di partire da un certo punto, ad esempio il punto A nella figura 16 e calcolare lì la

derivata della funzione. Ricordiamo che la derivata di una funzione in un certo punto è un numero che esprime quanto quella funzione "pende" in quel punto: ad esempio, in riferimento alla figura 16, la derivata nel punto A sarà un numero negativo piccolo per indicare che in A la funzione decresce di poco; nel punto B sarà un numero negativo grande, infatti in B la funzione è più ripida; nel punto C la derivata è nulla e in D è un numero positivo, per indicare che la funzione è crescente. La tecnica di discesa del gradiente consiste nel partire da un punto qualsiasi e fare uno spostamento inversamente proporzionale alla derivata in quel punto. Se ad esempio si partisse da B, essendo la derivata in B negativa, si farà uno spostamento positivo, avvicinandosi a C; se invece fossimo partiti da D, essendo la derivata positiva avremmo fatto uno spostamento all'indietro, avvicinandoci anche in questo caso a C. Ripetendo in ciclo il processo, il punto C viene raggiunto per piccoli passi e non più abbandonato, infatti in C la derivata è nulla, quindi lo spostamento è nullo. In conclusione, la formula per trovare il peso $W2_{kj}$ che dà errore minimo è la seguente:

$$\Delta W2_{kj} = -\varepsilon \frac{\partial E}{\partial W2_{kj}}$$

dove ε è una costante di proporzionalità chiamata *tasso di apprendimento*, e $\frac{\partial E}{\partial W2_{kj}}$ è la derivata rispetto al peso $W2_{kj}$ dell'errore commesso dalla rete durante la presentazione di un certo esempio.

Facendo alcuni passaggi matematici (riportati in appendice di questo capitolo), è facile sviluppare questa derivata, ottenendo:

$$\Delta W2_{kj} = \varepsilon \delta_j H_k$$

$$\delta_j = \left(D_j - Y_j\right) Y_j \left(1 - Y_j\right)$$

$\varepsilon =$ *valore qualsiasi consigliabile fra 0.1 e 0.9*

Queste sono le formule per l'apprendimento dei pesi W2, cioè quelli che vanno dal livello nascosto H al livello di output Y. Passiamo ora all'apprendimento dei pesi W1, cioè quelli delle connessioni del livello X col livello H. La tecnica di calcolo è la stessa, però vi è una difficoltà in più dovuta al fatto che dovremo considerare l'errore commesso dalle unità nascoste e questo non è immediato. L'idea di fondo è quella di propagare l'errore

all'indietro dal livello Y al livello H "pesandolo" coi valori W2 (prima della loro modifica). In altre parole, se l'unità Y_j ha commesso un certo errore, questo sarà stato proveniente da un errore commesso dalle unità H_k e propagato a Y_j attraverso i pesi $W2_{kj}$. Se ripercorro all'indietro i legami W2, sono in grado di ricostruire l'errore commesso da H_k. Traducendo questa idea in formule matematiche abbiamo:

$$\Delta W1_{ik} = -\varepsilon \frac{\partial E}{\partial W1_{ik}}$$

con passaggi matematici analoghi a quelli visti prima (riportati in appendice), otteniamo:

$$\Delta W1_{ik} = \varepsilon \delta_k X_i$$

$$\delta_k = \left(\sum_j \delta_j W2_{kj} \right) H_k (1 - H_k)$$

Si noti la ricorsività nella formula per il δ, che corrisponde alla propagazione all'indietro dell'errore. Per comodità, la sommatoria che compare nella formula di δ_k verrà chiamata *errore sull'unità H_k* e denotata con Err_H[k].

Riscriviamo dunque l'algoritmo di apprendimento "back propagation":

```
procedure apprendi;
  ripeti (* epoche *)
    per ogni esempio {X,D}
      esegui la rete con X e trova Y;
      backpropagate;
    fine;
    calcola errore dell'intera epoca;
  finché errore dell'epoca inferiore ad errore ammesso
fine;

procedure backpropagate;
  per ogni unità j del livello Y   (* aggiusta W2 e
                                     calcola Err_H *)
    delta := (D[j] - Y[j]) * Y[j] * (1 - Y[j]);
    per ogni unità k del livello H (compreso bias)
      Err_H[k] := Err_H[k] + (delta * W2[k,j]);
      W2[k,j]  := W2[k,j] + (epsilon * delta * H[k]);
    fine;
  fine;
```

```
    per ogni unità k del livello H  (* aggiusta W1 *)
      delta := Err_H[k] * H[k] * (1 - H[k]);
      per ogni unità i del livello X (compreso bias)
        W1[i,k] := W1[i,k] + (epsilon * delta * X[i]);
      fine;
    fine;
  fine;
```

Si noti un piccolo particolare: se all'inizio dell'apprendimento la rete avesse tutti i pesi W1 e W2 uguali a zero, la rete potrebbe non apprendere mai, infatti tutti gli errori Err_H[k] varrebbero zero e i pesi W1 non verrebbero mai modificati. È dunque indispensabile inizializzare tutti i pesi W1 e W2 con piccoli valori casuali.

Concludiamo questo capitolo con alcune considerazioni: questa tecnica di apprendimento ha il vantaggio di essere relativamente semplice e facile da implementare ma ha due grossi svantaggi. In primo luogo non garantisce la capacità di apprendere sempre, in secondo luogo è molto lento. Entrambe queste caratteristiche sono frutto del metodo "discesa del gradiente", infatti lo spostamento verso la discesa non garantisce di arrivare al minimo assoluto ma all'avvalamento più vicino, che potrebbe non essere il più basso di tutti. Inoltre il fatto di spostarsi di un fattore proporzionale alla pendenza fa si che i pendii più leggeri vengano percorsi "al rallentatore"; non vi è infatti alcuna tecnica di "rincorsa" o "inerzia" per raggiungere più velocemente i fondovalle. Esistono varie tecniche per accelerare e migliorare questo metodo di apprendimento, ma in questa sede non è il caso di parlarne.

Appendice

Sviluppo di $\Delta W2_{kj} = -\varepsilon \frac{\partial E}{\partial W2_{kj}}$

Sfruttiamo una nota regola matematica che afferma che la derivata di una funzione composta è calcolabile come prodotto di due altre derivate (composizione delle derivate): scomponiamo dunque la nostra derivata considerando il valore di attivazione A_j dell'unità j del livello Y. Otteniamo:

$$\Delta W2_{kj} = -\varepsilon \frac{\partial E}{\partial A_j} \frac{\partial A_j}{\partial W2_{kj}}$$

Per comodità di calcolo, introduciamo una nuova grandezza chiamata δ_j, avremo dunque:

$$\Delta W2_{kj} = \varepsilon \delta_j \frac{\partial A_j}{\partial W2_{kj}}$$

$$\delta_j = -\frac{\partial E}{\partial A_j}$$

La prima equazione si risolve facilmente ricordando che la formula di attivazione delle unità è:

$$A_j = \sum_k W2_{kj} H_k$$

quindi la sua derivata è:

$$\frac{\partial A_j}{\partial W2_{kj}} = H_k$$

L'equazione per il delta, invece, va risolta applicando una seconda volta la regola di composizione delle derivate, stavolta rispetto al valore dell'output Y_j:

$$\delta_j = -\frac{\partial E}{\partial Y_j} \frac{\partial Y_j}{\partial A_j}$$

Risolviamo queste due derivate una per volta: per la prima si consideri la definizione data dell'errore:

$$E = \frac{1}{2}\sum_j \left(D_j - Y_j\right)^2$$

con semplici passaggi si può calcolare la sua derivata rispetto a Y_j, ottenendo:

$$\frac{\partial E}{\partial Y_j} = -\left(D_j - Y_j\right)$$

Passiamo alla seconda derivata: ricordando la funzione di trasferimento T a forma di sigmoide:

$$Y_j = \frac{1}{1+e^{-A_j}}$$

si ottiene facilmente la sua derivata:

$$\frac{\partial Y_j}{\partial A_j} = Y_j\,(1-Y_j)$$

Ricucendo tutti i pezzi otteniamo le formule finali:

$$\Delta W2_{kj} = \varepsilon \delta_j H_k$$

$$\delta_j = \left(D_j - Y_j\right) Y_j\,(1 - Y_j)$$

$\varepsilon =$ *valore qualsiasi consigliabile fra 0.1 e 0.9*

Sviluppo di $\Delta W1_{ik} = -\varepsilon \frac{\partial E}{\partial W1_{ik}}$

Anche qui applichiamo la regola di composizione delle derivate:

$$\Delta W1_{ik} = -\varepsilon \frac{\partial E}{\partial A_k}\frac{\partial A_k}{\partial W1_{ik}}$$

la prima di queste due derivate la chiamiamo anche qui "delta"; la seconda derivata, invece, la risolviamo con la stessa tecnica adottata per W2, ottenendo:

$$\Delta W1_{ik} = \varepsilon \delta_k X_i$$

$$\delta_k = -\frac{\partial E}{\partial A_k}$$

Applichiamo ancora la regola di composizione delle derivate:

$$\delta_k = -\frac{\partial E}{\partial H_k}\frac{\partial H_k}{\partial A_k}$$

Il primo di questi due termini verrà chiamato *errore sull'unità H_k* e denotato con Err_H[k]; applichiamo ad esso la regola di composizione delle derivate, combinata con la regola della derivata della sommatoria:

$$\delta_k = -\left(\sum_j \frac{\partial E}{\partial A_j}\frac{\partial A_j}{\partial H_k}\right)\frac{\partial H_k}{\partial A_k}$$

Il primo termine nella sommatoria non è nuovo, infatti è già stato trovato durante il calcolo di W2, dove era stato battezzato "δ_j"; il secondo termine viene risolto ricordando la formula che dà l'attivazione delle unità, la cui derivata è semplicemente $W2_{kj}$; e terzo termine viene risolto con la stessa tecnica già vista per W2, ottenendo $H_k(1-H_k)$. Ricomponendo i vari pezzi si avrà:

$$\Delta W1_{ik} = \varepsilon \delta_k X_i$$

$$\delta_k = -\left(\sum_j \delta_j W2_{kj}\right) H_k (1-H_k)$$

7 Applicazione di una rete back propagation

Vediamo in questo capitolo un'applicazione semplice e significativa: una rete che apprende la capacità di lanciare in aria un sasso (argomento già visto nel capitolo 3). I parametri in gioco sono 4 (figura 17): velocità di lancio (Vel), angolazione di lancio (Ang), tempo di caduta (Tem) e gittata (Git = distanza fra il lanciatore e il punto di arrivo).

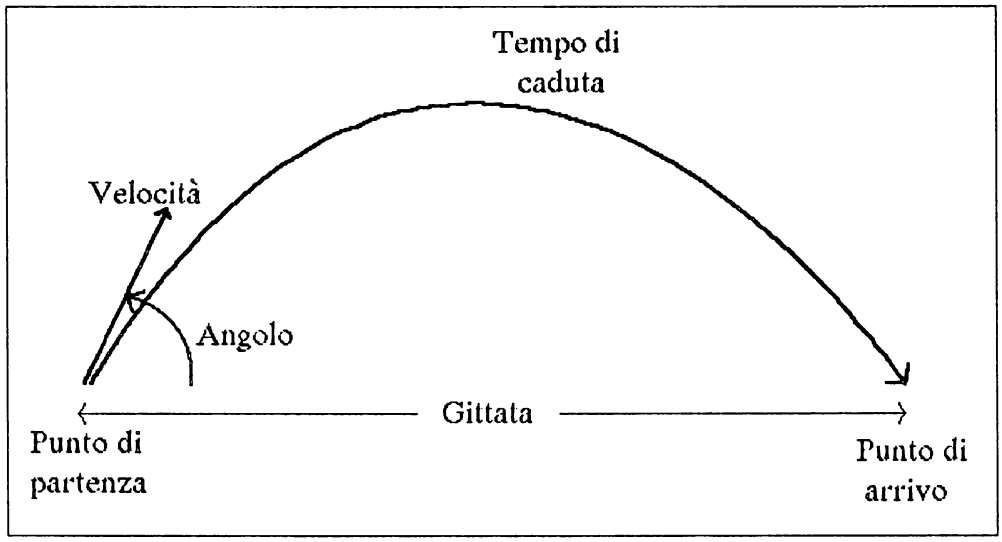

Figura 17. *I parametri in gioco nel lancio di un sasso.*

È noto che questo problema è risolubile con i classici metodi analitici della meccanica, che forniscono equazioni per mettere in relazione le quattro grandezze suddette; ad esempio, permettono di calcolare con esattezza la gittata a partire dalla velocità e dall'angolo di lancio, oppure il tempo di caduta a partire dalla gittata e dall'angolo, o qualsiasi altra combinazione.

Ci si può dunque chiedere perché scomodare le reti neurali per un problema di così consolidata risoluzione. In realtà quello che vogliamo evidenziare è il fatto che la capacità umana di lanciare oggetti in aria, e più in generale di muoversi nello spazio, non è affatto frutto di calcoli incosci che avvengono nel nostro cervello, ma frutto di addestramento ed allenamento per prove ed errori. Se qualcuno mi chiedesse di colpire un certo bersaglio con un sasso, non farei alcun calcolo ma tutt'al più qualche lancio di prova e poi tirerei senza indugio. Facciamo fare alla nostra rete neurale la stessa cosa.

Il fatto di aver scelto un campo applicativo di nota soluzione tradizionale comporta due vantaggi: innanzitutto la possibilità di verificare il risultato della rete confrontandolo col risultato ottenuto dalle equazioni, in secondo luogo la possibilità di procurarsi l'insieme degli esempi senza dover praticare fisicamente dei lanci ma simulando via software i lanci-prova. Prima di procedere, dunque, ricordiamo alcune formule utili:

$$Git = \frac{Vel^2 \sin(2Ang)}{g}$$

$$Tem = 2Vel\frac{\sin(Ang)}{g}$$

dove g è l'accelerazione di gravità (9.8 m/sec/sec). La posizione all'istante t del sasso può essere rappresentata mediante coordinate ortogonali R e C (Riga, Colonna):

$$R = Vel\cos(Ang)t$$

$$C = Vel\sin(Ang)t - \frac{1}{2}gt^2$$

L'istante di caduta a terra, assumendo di essere in pianura, sarà:

$$t = \frac{2Vel\sin(Ang)}{g}$$

Con queste formule è dunque banale fare un tracciamento sul video del computer della traiettoria di lancio, previo rapportamento di R e C alle dimensioni del video. È utile definire i limiti entro i quali spaziano le grandezze in gioco: assumiamo che la velocità di lancio sia compresa fra 0 e 100 metri al secondo e che l'angolazione sia compresa fra 0 e 90 gradi (che in radianti vale 3.14/2). Calcoliamo i limiti dell'area di lancio: l'altezza massima raggiungibile si otterrà lanciando il sasso con la velocità massima in verticale,

$$\text{Alt_max} = \frac{\text{Vel}^2 \sin(\text{Ang})^2}{2g} = \frac{100^2 \sin\left(\frac{3.14}{2}\right)^2}{2 \cdot 9.8} = \frac{10000}{2 \cdot 9.8}$$

la gittata massima si ha lanciando il sasso con velocità massima e angolazione pari a 45 gradi (3.14/4),

$$\text{Git_max} = \frac{\text{Vel}^2 \sin(2\text{Ang})}{g} = \frac{100^2 \sin\left(\frac{2 \cdot 3.14}{4}\right)}{9.8} = \frac{10000}{9.8}$$

il tempo massimo di volo si avrà lanciando il sasso con velocità massima in verticale,

$$\text{Tem_max} = \frac{2\text{Vel} \sin(\text{Ang})}{g} = \frac{2 \cdot 100 \sin\left(\frac{3.14}{2}\right)}{9.8} = \frac{200}{9.8}$$

Ma torniamo alla nostra rete neurale. Il suo scopo è quello di trovare quale velocità e quale angolazione imprimere al lancio di un sasso per farlo arrivare in un certo punto. Avremo dunque due unità di output della rete, che rappresentano velocità e angolazione, e due unità di input, che rappresentano gittata e tempo. Le unità nascoste della nostra rete possono essere scelte con vari criteri empirici, ma a priori è buona norma tentare con un numero di unità nascoste che superi il doppio delle unità di input, quindi impiegheremo 5 unità nascoste (figura 18).

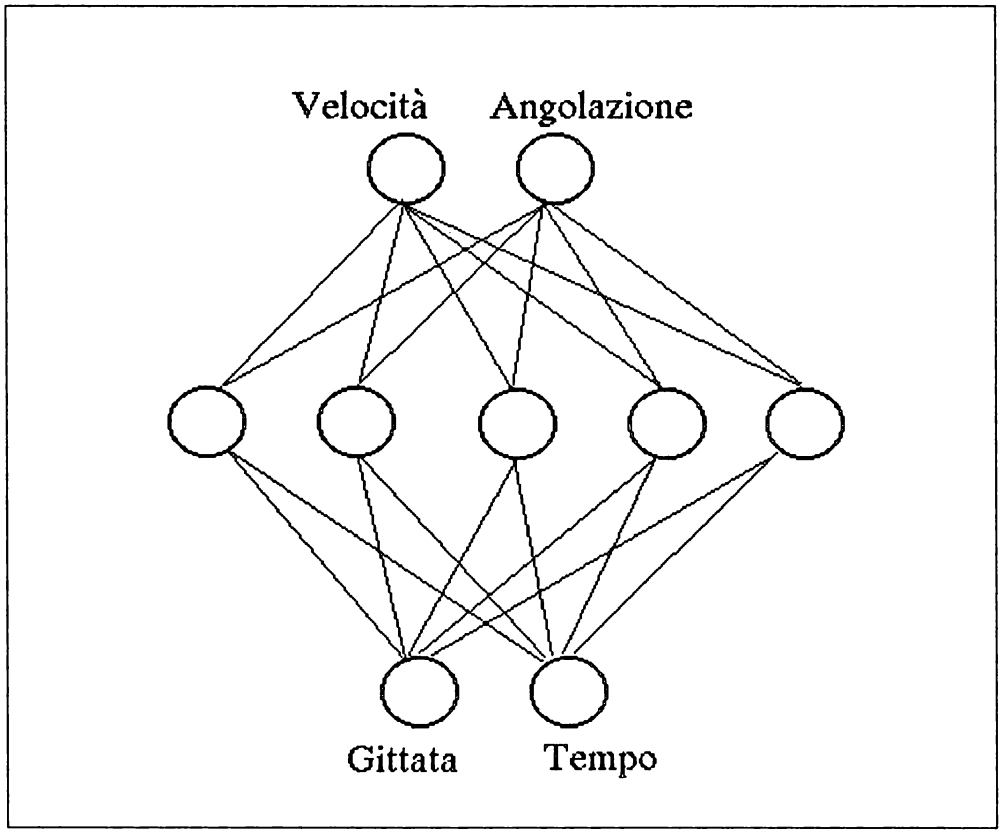

Figura 18. *Topologia della rete neurale per il lancio di un sasso.*

Si noti che l'unità di input che rappresenta il tempo di caduta è indispensabile, infatti a parità di gittata un tiro può essere teso o alto, e ciò che determina la forma della traiettoria è proprio il tempo di caduta, come evidenziato in figura 19.

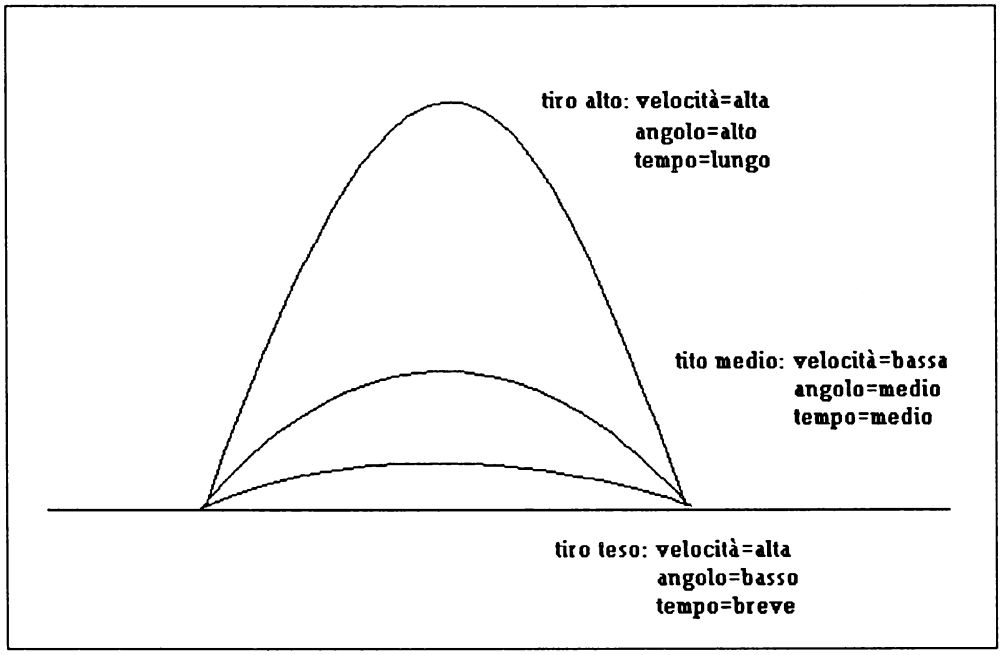

Figura 19. *Varietà di lanci a parità di gittata.*

Per loro natura, le unità di una rete back propagation possono assumere solo valori compresi fra 0 e 1, quindi è indispensabile che le grandezze in gioco siano rapportate con un fattore di scala all'intervallo [0..1]. Dato un qualsiasi lancio avremo dunque:

$$X_1 = \textit{gittata} = \frac{\text{Git}}{\text{Git_max}}$$

$$X_2 = \textit{tempo} = \frac{\text{Tem}}{\text{Tem_max}}$$

$$D_1 = \textit{velocità} = \frac{\text{Vel}}{100}$$

$$D_2 = \textit{angolo} = \frac{\text{Ang}}{\frac{3.14}{2}}$$

e, analogamente, dato un certo esito della rete avremo:

$$\textit{velocità} = Y_1 \cdot 100$$

$$\textit{angolo} = Y_2 \frac{3.14}{2}$$

L'addestramento della nostra rete avverrà generando un certo insieme di lanci-prova le cui grandezze saranno memorizzate in una lista di esempi chiamata E. Ogni esempio E[i] è una quaterna di numeri reali: E[i].Vel, E[i].Ang, E[i].Git, E[i].Tem. La costruzione di questi esempi avviene scegliendo a caso i valori Vel e Ang e calcolandosi i corrispettivi Git e Tem. Per evitare la generazione di lanci anomali, come ad esempio lanci orizzontali o lanci con velocità nulla, è meglio che il valore casuale di Vel sia compreso fra 30 e 100 e il valore di Ang sia compreso fra 30 e 80 gradi.

Una volta generati gli esempi, la nostra rete effettuerà un apprendimento per epoche in cui per ogni epoca vengono spazzolati tutti gli esempi. Per ogni esempio la rete viene eseguita e poi viene propagato all'indietro il suo eventuale errore. Durante questa fase viene tenuta traccia dell'errore commesso dalla unità Y_j che ha sbagliato di più (Err_rete, che in valore assoluto è compreso fra 0 e 1). Viene poi calcolato l'errore massimo commesso durante l'epoca (Err_epoca) trovando il massimo fra i vari Err_rete di ogni esempio. L'apprendimento continua a ciclare sulle epoche finché l'errore dell'epoca non scende al di sotto di un errore ammesso (Err_amm). A questo punto abbiamo la garanzia che la rete abbia appreso bene, cioè che per ogni esempio non vi sia alcuna unità che sbagli più di Err_amm. Ad esempio, supponendo di definire Err_amm=0.05, otterremo una rete in cui la velocità o l'angolazione del lancio non siano mai sbagliati più del 5% (il che non implica affatto che il sasso arrivi sul bersaglio con uno scarto del 5%). L'ottenimento di simili prestazioni può richiedere tempo di apprendimento estremamente lungo o addirittura infinito (nel caso di carenza di unità nascoste), quindi è necessario introdurre delle pause durante l'apprendimento con richiesta di conferma all'utente sul proseguimento.

Nel listato 3 compare il codice Pascal del programma SASSO, la cui versione eseguibile (SASSO.COM) è reperibile tramite le strutture divulgative di RAI-Televideo.

Listato 3:

```
(*-------------------------------------------------------------------
                    ESERCIZIO SU RETI NEURALI
                    MODELLO BACK PROPAGATION
                         PROGRAMMA SASSO
                       A. Mazzetti - 1991
-------------------------------------------------------------------*)

program sasso;
const max_X=51; max_H=51; max_Y=50; max_E=200;
type string12=string[12];
var X:array[1.. max_X]of real; (* unità di input + bias  *)
   H:array[1.. max_H]of real;  (* unità nascoste + bias *)
   Y:array[1.. max_Y]of real;  (* unità di output *)
   D:array[1.. max_Y]of real;  (* unità di output desiderato *)
   W1:array[1..max_X,1..max_H]of real;
                               (* matrice pesi 1^ livello *)
   W2:array[1..max_H,1..max_Y]of real;
                               (* matrice pesi 2^ livello *)
   NX,NH,NY,NE:integer;        (* numero unità, numero esempi *)
   epsilon, Err_rete:real;     (* tasso apprendimento, errore rete *)
   com:char;                   (* comando letto da tastiera *)
   i,k,j,p:integer;            (* indici vari *)
   Alt_max,Git_max,Tem_max:real;
                               (* altezza, gittata, tempo caduta *)
   E:array[1..max_E]of record Vel,Ang,Git,Tem:real end;
                               (* esempi *)


(*------------------------- back_propagate -------------------------*)
procedure back_propagate;
 var Err_H:array[1..max_H]of real; (* errore su unità nascoste *)
     delta:real;
 begin
   for k:=1 to NH+1 do Err_H[k]:=0.0; Err_rete:=0.0;
   for j:=1 to NY do begin
     if abs(D[j]-Y[j]) > Err_rete then Err_rete:=abs(D[j]-Y[j]);
     delta:=(D[j]-Y[j])*Y[j]*(1.0-Y[j]);
    for k:=1 to NH+1 do begin
       Err_H[k]:=Err_H[k]+(delta*W2[k,j]);
       W2[k,j]:=W2[k,j]+(epsilon*delta*H[k]);
     end;
```

```
    end;
    for k:=1 to NH do begin
      delta:=Err_H[k]*H[k]*(1.0-H[k]);
      for i:=1 to NX+1 do begin
        W1[i,k]:=W1[i,k]+(epsilon*delta*X[i]);
      end;
    end;
  end;
(*------------ funzione di trasferimento T a sigmoide -------------*)
function T(A:real):real;
  begin
    T:=1.0 / (1.0 + exp(-A) );
  end;
(*------------------------------ esegui ------------------------------*)
procedure esegui;
  var A:real;
  begin
    for k:=1 to NH do H[k]:=0.0;
    for j:=1 to NY do Y[j]:=0.0;
    for k:=1 to NH do begin
      A:=0.0;
      for i:=1 to NX+1 do A:=A+(W1[i,k]*X[i]);
      H[k]:=T(A);
    end;
    for j:=1 to NY do begin
      A:=0.0;
      for k:=1 to NH+1 do A:=A+(W2[k,j]*H[k]);
      Y[j]:=T(A);
    end;
  end;
(*------------ tracciamento traiettoria sul video ----------------*)
procedure traccia(Vel,Ang:real);
  var t:real; r,c:integer;
  begin
    t:=0.1; gotoxy(1,25);  (* posizionamento cursore sul video *)
    repeat
      c:=1+trunc(Vel*cos(Ang)*t*(80/Git_max));
      r:=25-trunc( ((Vel*sin(Ang)*t)-(0.5*9.8*t*t))*(25/Altmax));
      gotoxy(c,r); write('*');
      t:=t+0.1;
    until t>((2*Vel*sin(Ang))/9.8);
    gotoxy(1,1);write('premere RET ');readln;
    clrscr;   (* pulizia video *)
  end;
```

```
(*---------------- generazione esempi lancio sasso ----------------*)
procedure genera_esempi;
 begin
   clrscr;    (* pulizia video *)
   for p:=1 to NE do begin
     E[p].Vel:=(random(100)*(70/100))+30;
     E[p].Ang:=((random(90)*(50/90))+30)*(3.14/180);
     E[p].Git:=(E[p].Vel*E[p].Vel*sin(2*E[p].Ang))/9.8;
     E[p].Tem:=(E[p].Vel*2*sin(E[p].Ang))/9.8;
     traccia(E[p].Vel,E[p].Ang);
   end;
 end;

(*------------------------- apprendi -----------------------------*)
procedure apprendi;
 var Err_epoca,Err_amm:real; c:char; pausa,epoca:integer;
 begin
   write('Quanti esempi? (1..',max_E,'): ');
                  readln(NE);
   write('Errore massimo ammesso? (0.001..0.99): ');
                  readln(Err_amm);
   write('Tasso di apprendimento (epsilon)? (0.001..0.99): ');
                  readln(epsilon);
   write('Pausa ogni quante epoche? (0..10000): ');
                  readln(pausa);
   genera_esempi;
   epoca:=0;
   repeat  (* ciclo sulle epoche *)
     Err_epoca:=0.0;
    for p:=1 to NE do begin  (* ciclo sugli esempi *)
       X[1]:=E[p].Git/Git_max;  X[2]:=E[p].Tem/Tem_max;
       D[1]:=E[p].Vel/100;      D[2]:=E[p].Ang/(3.14/2);
       esegui;
       back_propagate;
       if Err_rete>Err_epoca then Err_epoca:=Err_rete;
     end;
     epoca:=epoca+1;
     writeln('epoca=',epoca:3,'  errore=',Err_epoca:7:5);
     if (epoca mod pausa)=0 then begin
       write('ancora ?(s/n) '); readln(c); if c='n' then
                                   Err_epoca:=-1.0;
     end;
   until Err_epoca<Err_amm
 end;
```

```
(*------------------------- modifica -------------------------*)
procedure modifica;
 begin
   write('Numero di unità' di input  (1..',max_X-1,') ? :
                                    ');readln(NX);
   write('Numero di unità' nascoste  (1..',max_H-1,') ? :
                                    ');readln(NH);
   write('Numero di unità' di output (1..',max_Y,') ? : ');
                                    readln(NY);
   for i:=1 to NX do X[i]:=0.0;  X[NX+1]:=1.0;
   for k:=1 to NH do H[k]:=0.0;  H[NH+1]:=1.0;
   for j:=1 to NY do Y[j]:=0.0;
   for i:=1 to NX+1 do
     for k:=1 to NH do
       W1[i,k]:=(random(100)-50.0)/100.0;
   for k:=1 to NH+1 do
     for j:=1 to NY do
       W2[k,j]:=(random(100)-50.0)/100.0;
   Alt_max:=10000/(2*9.8);
   Git_max:=10000/9.8;
   Tem_max:=200/9.8;
 end;

(*------------------------- visualizza -------------------------*)
procedure visualizza;
 var ch:char;
 begin
   writeln; writeln('unità' di input : ');
   for i:=1 to NX+1 do write(X[i]:4:2,' '); writeln;
   writeln('unità' nascoste : ');
   for k:=1 to NH+1 do write(H[k]:4:2,' '); writeln;
   writeln('unità' di output: ');
   for j:=1 to NY do write(Y[j]:4:2,' '); writeln;
   writeln; write('Vuoi vedere matrice W1 (s/n) ? : ');readln(ch);
   if ch='s' then begin
     write(' ':4);
     for k:=1 to NH do write(k:2,'H   '); writeln;writeln;
     for i:=1 to NX+1 do begin
       write(i:2,'X ');
       for k:=1 to NH do write(W1[i,k]:5:2,' '); writeln;
     end;
     writeln;
   end;
   write('Vuoi vedere matrice W2 (s/n) ? : ');readln(ch);
   if ch='s' then begin
```

```
    write(' ':4);
    for j:=1 to NY do write(j:2,'Y   '); writeln;writeln;
    for k:=1 to NH+1 do begin
      write(k:2,'H ');
      for j:=1 to NY do write(W2[k,j]:5:2,' '); writeln;
    end;
    writeln;
  end;
end;


(*------------------------- salva ------------------------------*)
procedure salva;
 var f:text; fn:string12;
 begin
   write('Nome del file su cui salvare i pesi: ');readln(fn);
   assign(f,fn); rewrite(f);
   writeln(f,NX);writeln(f,NH);writeln(f,NY);
   for k:=1 to NH do
     for i:=1 to NX+1 do
       writeln(f,W1[i,k]:7:3);
   for j:=1 to NY do
     for k:=1 to NH+1 do
       writeln(f,W2[k,j]:7:3);
   close(f);
 end;


(*------------------------- carica ------------------------------*)
procedure carica;
 var f:text; fn:string12; r:integer;
 begin
   write('Nome del file contenente i pesi: ');readln(fn);
   assign(f,fn); reset(f);
   readln(f,NX);readln(f,NH);readln(f,NY);
   for k:=1 to NH do
     for i:=1 to NX+1 do
       readln(f,W1[i,k]);
   for j:=1 to NY do
     for k:=1 to NH+1 do
       readln(f,W2[k,j]);
   close(f);
 end;
```

```
(*------------------------- definisci_lancio -------------------------*)
procedure definisci_lancio;
 var c:integer; ch:char; t:real;
 begin
   clrscr;  (* pulisci video *)
   write('Posizionare il bersaglio coi tasti freccia, poi predere
                                        RET');
   c:=40; gotoxy(c-1,25); write('—');
   repeat
     read(kbd,ch); if ch=chr(27) then read(kbd,ch);
                                        (*leggi tasti freccia *)
     gotoxy(c-1,25); write('   ');
     if ch=chr(75) then if c>3  then c:=c-1;
                                        (* freccia verso sinistra *)
     if ch=chr(77) then if c<75 then c:=c+1;
                                        (* freccia verso destra *)
     gotoxy(c-1,25); write('—-');
   until ch=chr(13);
   X[1]:=c/80;
   gotoxy(1,1); for c:=1 to 80 do write(' '); gotoxy(1,1);
   write('Tempo di caduta? (1..',Tem_max:2:0,' sec.):
                                        ');readln(t);
   gotoxy(1,1); for c:=1 to 80 do write(' '); gotoxy(1,1);
   X[2]:=t/Tem_max;
 end;
(*------------------------------- main -------------------------------*)
begin
 modifica;
 repeat
   writeln; write('Apprendi Esegui Carica Salva Visualizza
                                        Modifica Fine: ');
   readln(com);
   case com of
   'a':apprendi;
   'e':begin definisci_lancio; esegui; traccia(Y[1]*100,Y[2]*
                                        (3.14/2)) end;
   'c':carica;
   's':salva;
   'v':visualizza;
   'm':modifica;
   end;
 until com='f'
end.
```

Listato 3: *Codice Pascal del programma SASSO, per la prova del modello back propagation.*

L'interfaccia di questo programma è molto simile a quella dei programmi MEM_ASS e IMMAGINI già visti in precedenza. Lanciando il programma SASSO si ottiene un menù di questo tipo:

```
Apprendi Esegui Carica Salva Visualizza Modifica Fine:
```

il cui significato delle voci è lo stesso degli altri programmi. Il comando "a" (Apprendi) comporta l'acquisizione dei parametri da usare durante l'apprendimento: numero di esempi, errore ammesso, tasso di apprendimento e collocazione della pausa. Segue la visualizzazione degli esempi autogenerati casualmente e poi la fase di apprendimento vera e propria in cui per ogni epoca viene visualizzato sul video l'errore; l'apprendimento termina o quando viene raggiunto l'errore ammesso o quando l'utente interrompe durante una pausa. Segue una traccia dell'esecuzione:

```
C:> SASSO
Numero di unità di input  (1..50) ? : 2
Numero di unità nascoste  (1..50) ? : 5
Numero di unità di output (1..50) ? : 2
Apprendi Esegui Carica Salva Visualizza Modifica Fine: a

Quanti esempi? (1..200): 200
Errore massimo ammesso? (0.001..0.99): 0.02
Tasso di apprendimento (epsilon)? (0.001..0.99): 0.3
Pausa ogni quante epoche? (0..10000): 100
```

```
premere RET



        *********
     ****       ****
    ***           ***
  ***               ***
 **                   **
*                       *
```

```
premere RET



        ****
       **  **
      **    **
     **      **
     **      **
    **        **
    *          *
   **          **
   *            *
   *            *
  *              *
```

```
...     (fa vedere 200 lanci-prova casuali)

epoca=1  errore=0.34382
epoca=2  errore=0.35986
epoca=3  errore=0.35977

...

epoca=100  errore=0.27663
ancora? (s/n): s
epoca=101  errore=0.27019

...

epoca=5800  errore=0.02103
ancora? (s/n): n

Apprendi Esegui Carica Salva Visualizza Modifica Fine: _
```

Si noti che può capitare che all'inizio l'errore dell'epoca aumenti anziché diminuire: questo è dovuto ad una sorta di "confusione" in cui la rete incappa all'inizio, cioè quando i pesi sono ancora molto casuali; poi, man mano che i pesi vengono modificati, si orientano verso i minimi d'errore e continuano la loro discesa asintotica. Può capitare che l'errore zero non venga mai raggiunto in tempi accettabili (polinomiali).

Nel listato 3 si può osservare la procedura "apprendi" in cui compare la generazione degli esempi seguita dal ciclo sulle epoche contenente il ciclo sugli esempi; per ogni esempio vi è l'assegnamento delle unità di input X e di output desiderato D, seguito dall'esecuzione (in cui vengono calcolate le unità Y) e dalla backpropagation (che aggiusta i pesi W1 e W2). Si noti l'immagazzinamento di "Err_epoca" (errore della rete nell'esempio che ha sbagliato di più) e la terminazione del ciclo sulla base dell'errore.

Nella procedura "genera_esempi" compare la generazione casuale della velocità e angolazione, seguita dal calcolo della gittata e del tempo, a sua volta seguita dal tracciamento sul video dalla traiettoria di volo. La procedura "traccia" contiene un ciclo in cui ad ogni istante di tempo si calcolano le coordinate del punto in cui è posizionato il sasso e lo si visualizza col carattere asterisco. La procedura "esegui" contiene l'azzeramento iniziale delle unità nascoste e di output, seguito dai due cicli: uno per passare dal livello X a quello H, l'altro per passare da H a Y. Si noti la funzione di trasferimento "T" a sigmoide.

La procedura "back_propagate" contiene anch'essa il doppio ciclo, ma in ordine inverso: il primo per propagare l'errore da Y ad H, aggiornando i pesi W2; l'altro per propagare l'errore sulle unità nascoste (Err_H[k]) da H a X, aggiornando i pesi W1. Si noti che Err_H[k] viene calcolato durante il primo ciclo e usato nel secondo ciclo; si noti anche l'immagazzinamento di Err_rete (valore assoluto dell'errore dell'unità Y che ha sbagliato di più).

Il comando del menù principale "e" (Esegui) permette di provare la rete facendole fare un tiro verso un dato bersaglio. Il posizionamento del bersaglio viene fatto in modo grafico dalla procedura "definisci_lancio": il bersaglio è rappresentato da tre trattini "—" che si possono muovere a destra e sinistra mediante i tasti freccia. Segue la richiesta del tempo di caduta desiderato e l'assegnamento delle unità di input della rete X[1] e X[2] con valori opportunamente normalizzati. Terminata la procedura "definisci_lancio", segue il richiamo della procedura "esegui", che calcola i valori di Y[1] e Y[2], a sua volta seguita dalla procedura "traccia" a cui vengono passati come parametri i valori Y[1] e Y[2] opportunamente de-normalizzati. Ecco una traccia dell'esecuzione:

```
Apprendi Esegui Carica Salva Visualizza Modifica Fine: e
Posizionare il bersaglio coi tasti freccia, poi premere RET
    ...
```

```
Tempo di caduta? (1..20 sec.): 12

                 ****
            *****************
        ******            ******
    ****                      ****
  ***
**                              —
```

```
Apprendi Esegui Carica Salva Visualizza Modifica Fine: v
unità di input :
0.50 0.59 1.00
unità nascoste :
0.80 0.11 0.07 0.01 0.98 1.00
unità di output:
0.73 0.61
Vuoi vedere matrice W1 (s/n) ? : s
      1H    2H    3H    4H    5H
 1X -4.22 -0.15  0.86 -8.61  4.11
 2X  2.50  5.49 -6.35  1.33  4.28
 3X  2.02 -5.26  0.68 -0.87 -0.52

Vuoi vedere matrice W2 (s/n) ? : s
      1Y    2Y
 1H -4.15  1.61
 2H  6.87  1.58
 3H  1.55 -5.78
 4H  1.35  6.64
 5H  4.94  0.18
 6H -1.41 -0.90
Apprendi Esegui Carica Salva Visualizza Modifica Fine: f
C:>
```

Si noti il comando "v" (Visualizza) per vedere i valori delle unità dei livelli X, H e Y seguiti dai valori delle matrici W1 e W2. I comandi "c" (Carica) e "s" (Salva) caricano e salvano i pesi in maniera identica a quella già vista per le memorie associative. Il comando "m" (Modifica) acquisisce il numero di unità dei tre livelli e poi inizializza le matrici W1 e W2 con valori casuali compresi fra +0.5 e -0.5 (per ovviare ai problemi di non apprendibilità per simmetria); seguono le inizializzazioni dei valori massimi di altezza, gittata e tempo.

L'utilizzo di questo programma potrà dare una sensazione dell'estrema onerosità della fase di apprendimento e dell'estrema snellezza della fase di esecuzione: per dare un'idea di massima, l'esecuzione della rete, cioè la fase che inizia dopo l'acquisizione dall'utente del tempo di caduta e finisce quando parte il sasso, dura una frazione impercettibile di secondo. La fase di apprendimento, invece, avendo 200 esempi e un tasso di apprendimento di 0.3, arriverà ad un errore inferiore a 0.03 dopo qualche migliaio di epoche; su un PC AT 286 questo può richiedere una ventina di ore di elaborazione (il file SASSO.W contiene i pesi appresi in questo modo, vedi figura 20).

Si noti che la figura 20 esprime la "conoscenza" che la rete ha sul problema del lancio del sasso; in altri termini, questo guazzabuglio di numeri incomprensibili è l'equivalente di tutte le formule matematiche viste prima a proposito della velocità, gittata, tempo ed angolazione del lancio; appare sensazionale, dunque, la compattezza dell'informazione contenuta in una rete neurale. Per contro può sgomentare il fatto che la conoscenza contenuta in quella rete sia del tutto incomprensibile all'uomo: ad esempio, in quella trentina di numeri è impressa la conoscenza relativa al fatto che per ottenere un lancio lento non bisogna tirare lentamente ma con forza in alto. La conoscenza contenuta in questa rete è difficilmente dominabile da parte umana: se ad esempio si scoprisse che la rete sbaglia in certi casi, avremmo serie difficoltà nell'individuare il punto d'errore ed effettuare correzioni.

Esistono alcune tecniche per indagare nel contenuto cognitivo di una rete neurale, ma si tratta per lo più di tentativi artigianali. Uno dei metodi più usati è l'estrazione di *mappe di attività* delle unità nascoste: si tratta di rappresentazioni grafiche che evidenziano il valore di output di una certa unità H_k al variare dei

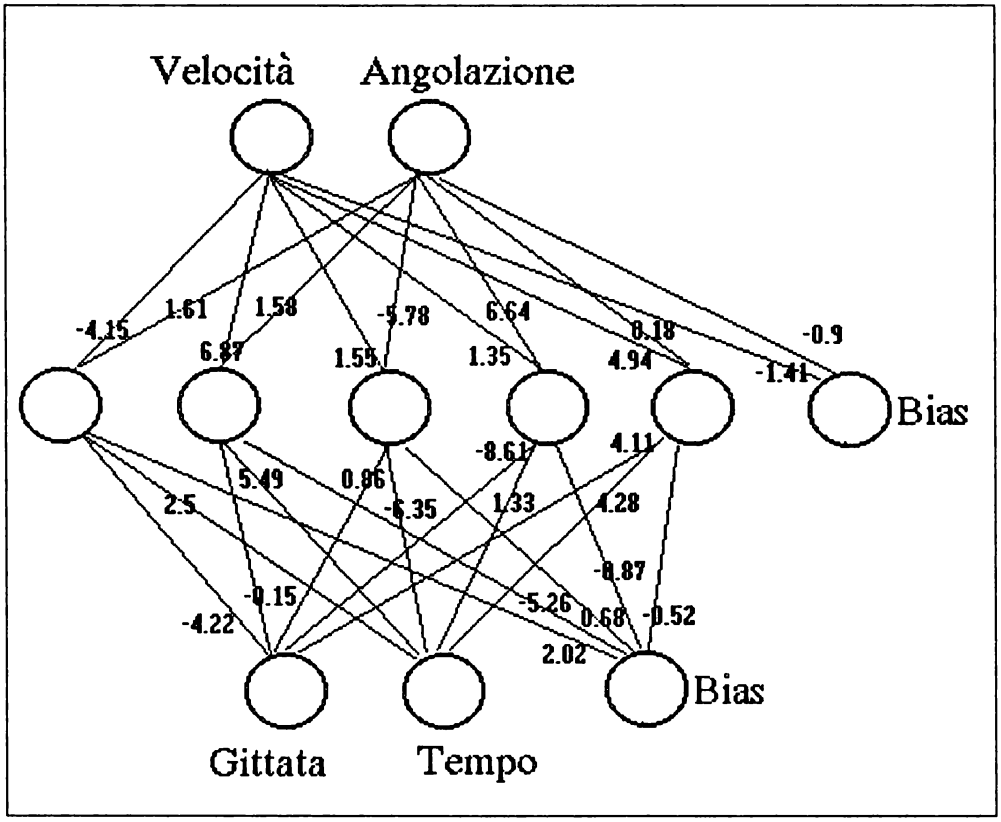

Figura 20. *La conoscenza appresa dalla rete, relativa al lancio del sasso.*

valori delle unità di input. Nel nostro caso, avendo due unità di input, otterremo una superficie in uno spazio euclideo tridimensionale (figura 21).

Sviluppando una mappa di attività per ogni unità nascosta otterremo una sensazione del loro ruolo nell'elaborazione; nel nostro caso, ad esempio, si può individuare quale unità ha a che fare con i lanci alti e quale con quelli tesi. Tuttavia questa informazione è poco utile dal punto di vista pratico perché non consente alcuna manipolazione della conoscenza della rete. L'unico modo di modificare la conoscenza della rete è un riaddestramento effettuato con un più accurato insieme di esempi.

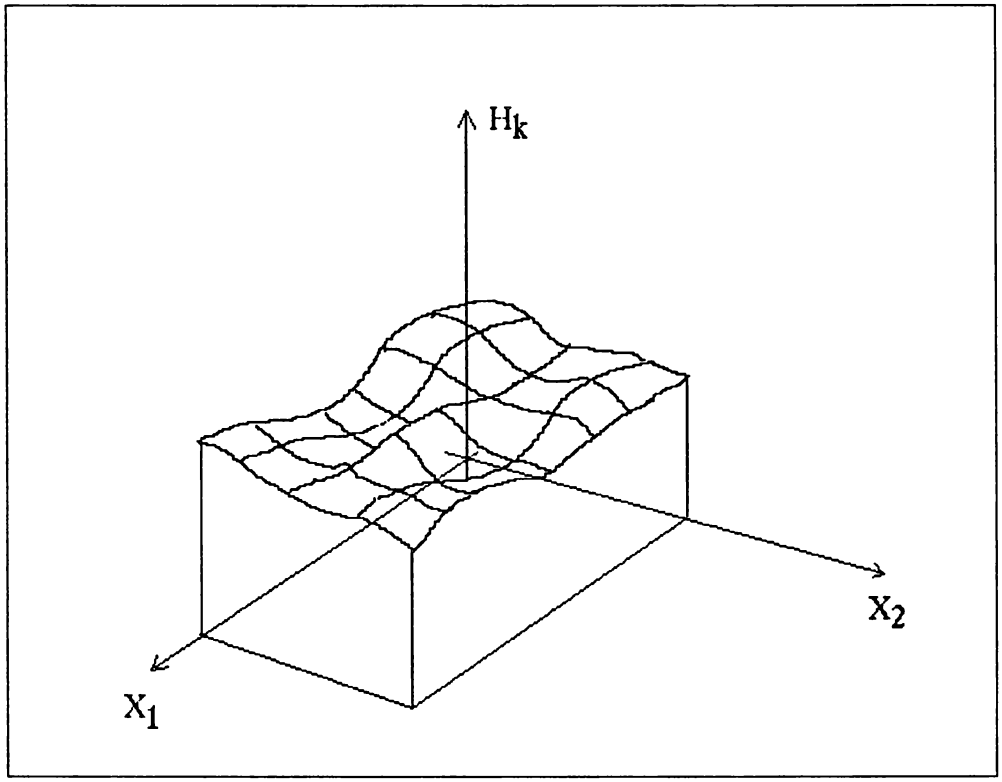

Figura 21. *Mappa di attività di una unità nascosta.*

8 Altri modelli di rete neurale

I modelli di rete visti finora (memoria associativa e back propagation) sono i più significativi ma non esauriscono tutte le possibilità. Ci proponiamo in questo capitolo di indagare altri modelli, senza entrare troppo in dettaglio. Analizzeremo tre modelli molto differenti fra loro e capostipiti di tre interessanti filoni di ricerca: modelli stocastici, modelli auto-organizzanti e modelli genetici.

Modelli stocastici

Nell'ambito dei modelli stocastici l'architettura più nota è la cosiddetta *Macchina di Boltzmann* [18] il cui nome manifesta l'analogia coi sistemi termodinamici. Si può notare infatti una corrispondenza fra l'attivazione delle unità di una rete neurale e la posizione delle molecole di un materiale metallico; l'organizzazione a reticolo cristallino impone un certo equilibrio delle forze interagenti fra le diverse molecole, proprio come nelle reti neurali le connessioni fra le unità implicano una certa relazione di "buon vicinato".

È noto però che nei metalli le molecole non sono ferme ma in continua agitazione termica: quanto più è elevata la temperatura, tanto più le molecole sono ribelli alla loro posizione; l'agitazione termica è assimilabile ad uno spostamento casuale rispetto alla posizione di equilibrio. Esiste una nota tecnica metallurgica chiamata *tempra* basata su un trattamento termico allo scopo di forzare ogni molecola nella miglior posizione, per ottenere l'indurimento del materiale. Scaldando un materiale

metallico, infatti, aumentiamo la casualità della posizione delle molecole; raffreddandolo molto lentamente l'agitazione delle molecole si riduce man mano fino ad annullarsi (congelamento). Le molecole si fermano (o quasi) nella posizione in cui si danno meno fastidio fra di loro, vale a dire nella posizione di equilibrio delle forze reciproche.

Ma torniamo alle reti neurali: anche qui il problema è quello di tovare la migliore forma di convivenza delle unità, cioè la migliore combinazione dei valori di attività. Introducendo nelle reti neurali il concetto di temperatura, inteso come agitazione casuale ribelle delle unità, possiamo effettuare delle operazioni di *tempra simulata* (simulated annealing [19]) allo scopo di ottenere situazioni di equilibrio ottimale. L'architettura che più si presta a questa estensione è quella della memoria associativa, opportunamente riveduta e corretta. Consideriamo un insieme di unità a valori binari {0,1}, totalmente connesse e organizzate in modo che alcune di esse rappresentino l'input/output della rete e altre siano nascoste, come in figura 22.

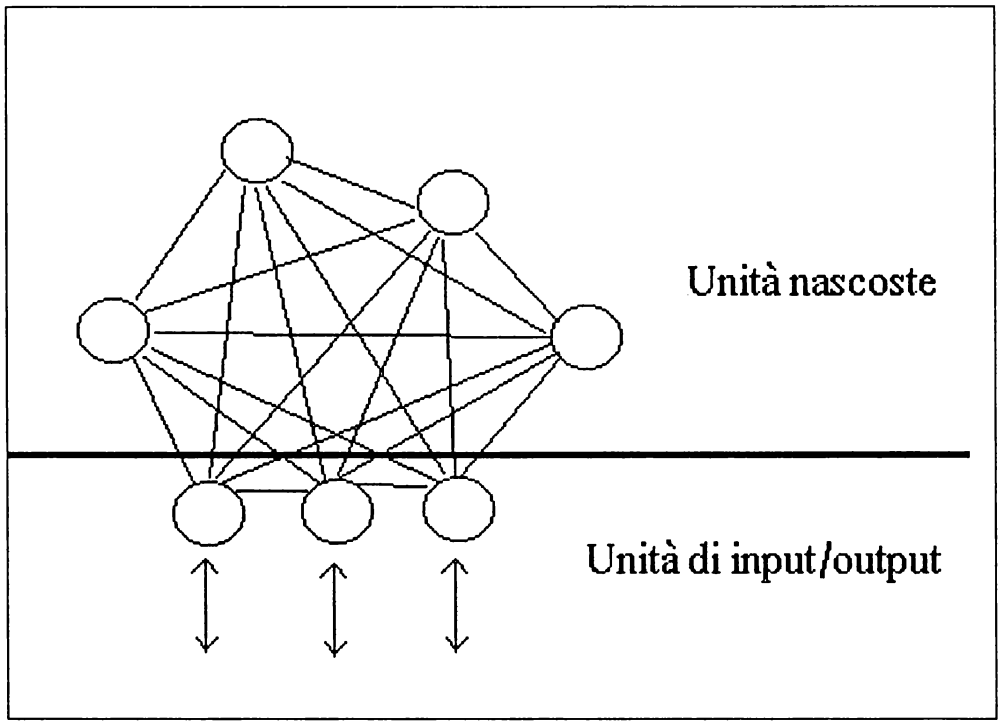

Figura 22. *Architettura di una macchina di Boltzmann*

La funzione di trasferimento T non è più di tipo deterministico ma probabilistico, nel senso che una unità ha una certa probabilità di non commutare il suo stato anche quando dovrebbe. Questa forma di ribellione è tanto più spinta quanto più è alto il parametro temperatura. Una formulazione matematica di questo meccanismo può essere:

$$P(X_i = 1) = \frac{1}{1 + e^{-\frac{A_i}{Temp}}}$$

$$A_i = \sum_j W_{ij} X_j$$

dove $P(X_i = 1)$ indica la probabilità che X_i valga 1. Si noti che facendo tendere la temperatura a zero, la funzione probabilità tende al gradino (già visto nel capitolo delle memorie associative), quindi il comportamento delle unità non è ribelle ma conformista. All'aumentare della temperatura la funzione assume una forma a sigmoide (simile a quella vista nei modelli back propagation, ma usata qui per scopi del tutto diversi); ad esempio, se A_i valesse +10, con una temperatura di +9 avremmo una probabilità del 75% circa che l'i-esima unità valga 1 e una probalità del 25% circa che valga 0. Con una scelta a caso nell'ambito di questa distribuzione di probabilità , l'i-esima unità assume il suo valore. Come si può notare, l'unità sarà ribelle nel 25% dei casi. Alle alte temperature (Temp tendente ad infinito) la probabilità si appiattisce sul valore 0.5, quindi il comportamento delle unità è totalmente casuale. In figura 23 compare il grafico della funzione probabilità al variare della temperatura.

Lo scopo di tutto questo meccanismo è di far evolvere la rete fino al minimo globale di energia, senza il pericolo di incepparsi in minimi locali (vi ricordate, nel capitolo sulle memorie associative, la "superficie di energia" assimilabile ad un foglio di gomma con crateri?). Un'utile metafora per la comprensione del meccanismo di tempra simulata può essere quello di rappresentare lo stato della rete come una pallina su una superficie montagnosa (non di gomma stavolta) in cui la temperatura si manifesti come vibrazione del terreno. Quanto più forte è la vibrazione, tanto più la pallina salterà (contrastando la forza di gravità); riducendo la vibrazione, la pallina si assesterà nella

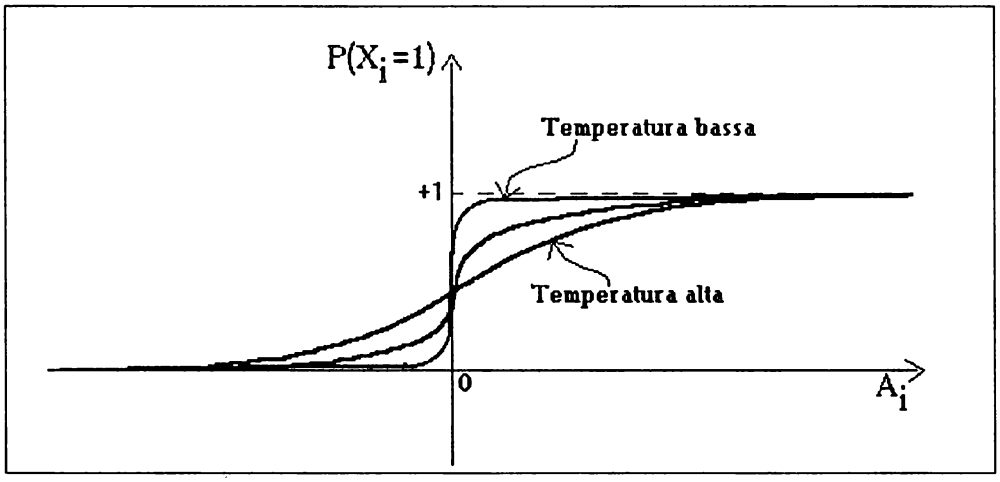

Figura 23. *La funzione di probabilità delle unità per diversi valori della temperatura.*

conca più larga e/o più profonda; al cessare della vibrazione la pallina rotolerà nel fondovalle.

Le reti neurali stocastiche hanno il vantaggio di trovare la soluzione migliore con alta probabilità e lo svantaggio di essere lente e computazionalmente onerose. La plausibilità biologica di questo tipo di rete è fortemente in dubbio fra gli esperti.

Il meccanismo di apprendimento per questo tipo di rete è più complesso di quelli visti finora, fermo restando un'impronta di tipo Hebbiano, a cui va aggiunta una componente stocastica. Fondamentalmente l'apprendimento avviene in due fasi: nella prima la rete viene congelata più volte per ogni esempio avendo assegnate le unità di input/output. I pesi vengono inizialmente modificati secondo la legge:

$$\Delta W_{ij} = \varepsilon X_i X_j$$

Nella seconda fase la rete viene congelata altrettante volte lasciando libere le unità di input/output e i pesi vengono modificati secondo una legge anti-Hebbiana:

$$\Delta W_{ij} = -\varepsilon X_i X_j$$

In entrambe le fasi la rete registra il comportamento delle unità visibili dopo che sia stato raggiunto l'equilibrio termico e calcola le relative statistiche: la rete avrà appreso quando le distribuzioni di probabilità coincidono approssimativamente.

Sono state suggerite singolari analogie fra la legge anti-Hebbiana di apprendimento e la funzione del sonno REM degli esseri viventi [20] in cui vengono "disimparate" informazioni parassite.

Le macchine di Boltzmann possono essere impiegate come memorie associative in cui a fronte di un input incompleto o errato viene fornita mediante congelamento la versione completa o corretta più verosimile. La presenza di unità nascoste e di meccanismi stocastici permette di superare i problemi tipici dei modelli di Hopfield e di Kosko visti nei capitoli precedenti a proposito delle memorie associative.

Modelli auto-organizzanti

Passiamo ora ad un modello di rete neurale totalmente differente, noto come *rete auto-organizzante*. Lo scopo di questo tipo di rete non è più quello di fornire un output a fronte di un certo input, ma quello di ricevere solo alcuni input e classificarli. In altre parole, la rete organizza in modo autonomo una struttura che accomuna gli input simili fra loro individuando regolarità e somiglianze. Il tipo di apprendimento di queste reti è detto *non supervisionato* perché gli esempi non contengono l'output, quindi non vi è informazione su cosa farsene degl'input. Lo scopo della rete non è quello di apprendere una relazione fra input e output ma quello di "osservare" un certo insieme di esempi e individuare possibili *categorie*.

Ad esempio, si consideri il caso in cui i dati di input siano matrici di pixel che rappresentano caratteri; nel capitolo sulle memorie associative abbiamo visto un'applicazione in cui la rete aveva il compito di associare un carattere minuscolo al suo corrispondente maiuscolo. Qui invece forniamo soltanto i caratteri minuscoli alla rete e le chiediamo di classificarli; la rete potrà trovare alcune interessanti categorie come ad esempio: i caratteri *tondeggianti* (b,d,g,o,p,q...), i caratteri *con la stanghetta* (l,t,b,d,p,q...), i caratteri *con le gambe* (m,n,i...). In altre parole, la rete raggruppa gli esempi simili; una volta apprese queste categorie, la rete potrà essere usata per individuare la categoria di

appartenenza di un certo carattere, anche mai visto, ad esempio, il carattere alfa potrà essere collocato fra i tondeggianti.

Esistono varie topologie per questo modello di rete, la più semplice e significativa è stata studiata da Kohonen [21] ed è rappresentata in figura 24.

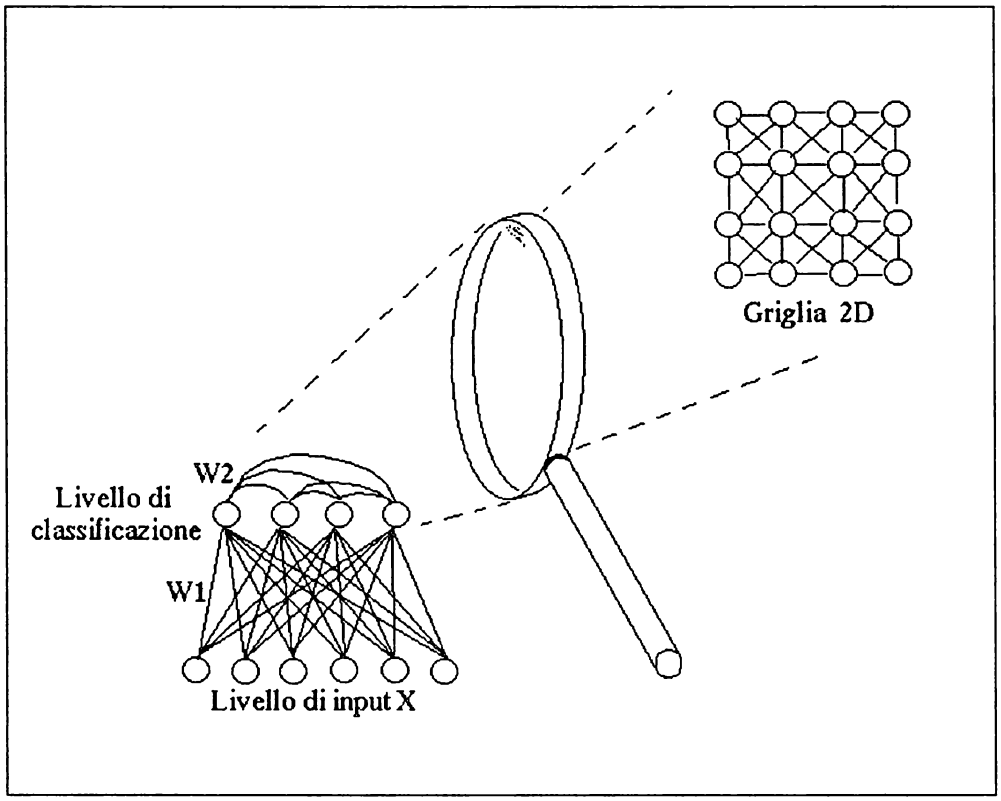

Figura 24. *Architettura di una rete auto-organizzante*

Vi sono due livelli, chiamati X e Y, di cui il primo è quello di input e il secondo è quello che rappresenta la classificazione. Vi sono due tipi di connessioni: quelle dal livello X al livello Y, chiamate W1, e quelle fra le unità del livello Y, chiamate W2, che (nella versione più semplice) non sono totali ma organizzate in uno schema a griglia in due dimensioni (ogni unità è connessa solo con le sue "vicine"). Queste ultime connessioni, W2, non vengono apprese ma sono fisse ed hanno una struttura a "cappello messicano", vale a dire che per ogni unità sono positive verso le unità limitrofe e negative verso le unità nell'anello adia-

cente. Ogni unità del livello Y riceve dunque due stimoli: uno dal livello X e l'altro dai vicini di casa:

$$A_j = \sum_i W1_{ij} X_i + \sum_k W2_{kj} Y_k$$

Lo stimolo reciproco fra le unità del livello Y crea un meccanismo di competizione che porta ciclicamente alla supremazia di una singola unità. Infatti, avendo ogni unità connessioni negative con quelle limitrofe, si scatena un processo di inibizione reciproca in cui chi è inizialmente più forte inibisce di più gli altri e viene inibito di meno, quindi sopravvive (ha un'attivazione elevata) facendo soccombere gli altri (attivazione bassa). Per questo motivo tale tipo di rete viene anche detto *rete competitiva* [22]. L'esecuzione di questa rete avviene in questo modo: dato un input qualsiasi X, la rete propaga verso il livello Y il segnale facendo "accendere" l'unità Y_j che rappresenta la classe di appartenenza dell'input dato. Qualora l'input sia ambiguo, cioè a metà strada fra più classi, il meccanismo di competizione fa sì che si formi una "bolla" di unità vincenti.

L'apprendimento di questa rete avviene in questo modo (ricordiamo che solo i pesi W1 vanno appresi, i pesi W2 sono fissi e a cappello messicano): data una distribuzione inizialmente casuale dei pesi W1, per ogni nuovo esempio X viene eseguita la rete e tovata la bolla vincente; poi vengono modificati solo i pesi delle unità interne alla bolla. Supponiamo per semplicità che la bolla sia composta da una sola unità vincente Y_v, verranno quindi modificati solo i pesi $W1_{iv}$, rappresentati in figura 25.

La variazione dei pesi è congegnata in modo da accentuare il fatto che l'input X appartenga alla classe individuata (quella rappresentata da Y_v); in altre parole la rete rafforza la sua convinzione che l'input X debba appartenere alla classe Y_v. Man mano che la rete esamina gli esempi, si autoconvince sempre più che gli esempi simili debbano far vincere la stessa unità, quindi che appartengano alla stessa classe. La formula è la seguente (nella sua forma più semplice):

$$\Delta W1_{iv} = \varepsilon\left(X_i - W1_{iv}\right)$$

dove v è l'indice dell'unità vincente (o l'insieme degli indici delle unità interne alla bolla vincente). I pesi afferenti alle unità perdenti non vengono modificati. Alla fine della fase di appren-

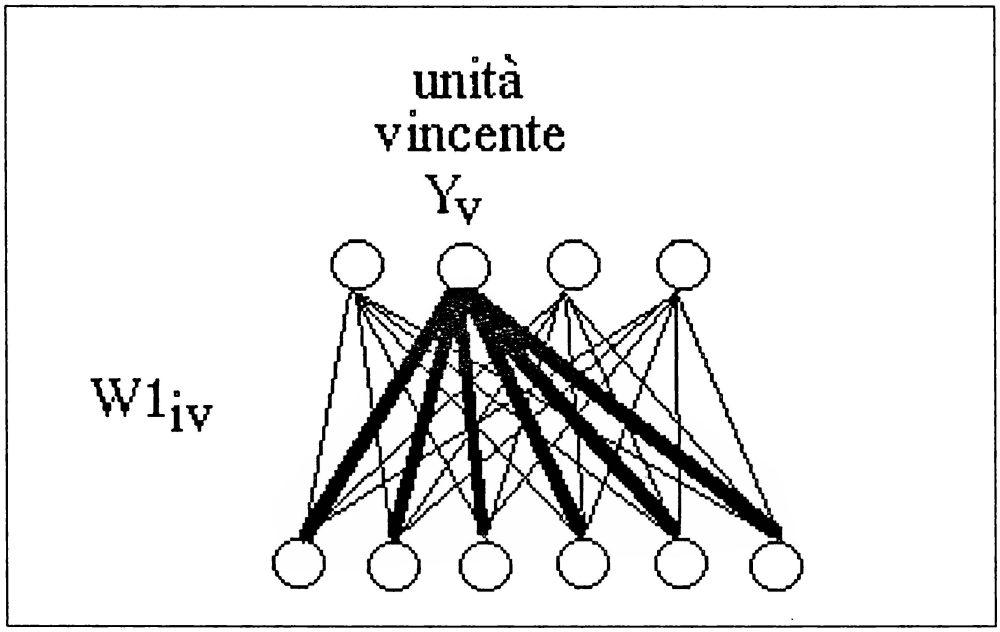

Figura 25. *I pesi soggetti a modifica durante l'esame di un dato esempio.*

dimento la configurazione dei pesi W1 rappresenta una quantizzazione dello spazio degli esempi ricevuti, vale a dire una partizione che raggruppa gli esempi simili fra loro, ove il concetto di somiglianza dipende dalla particolare distribuzione degli esempi analizzati.

Applicazioni di questo tipo di rete hanno avuto successo nel campo dell'analisi del segnale vocale e nel campo della classificazione di immagini. Sono particolarmente indicate quelle applicazioni in cui non si sappia a priori come elaborare certe informazioni, quindi ci si aspetta che la rete "scopra" da sola concetti interessanti in una popolazione di dati non strutturati [23]. La plausibilità biologica di questo modello è molto accreditata a causa della presenza di strutture auto-organizzanti nella corteccia cerebrale [24].

Modelli genetici

Dalla teoria dell'evoluzione di Darwin si è derivato un modello di rete in cui l'apprendimento è frutto di selezione naturale in una popolazione di individui che si riproducono [25, 26]. Si supponga di voler far apprendere ad una rete neurale un certo insieme di esempi: anziché definire una "formula di apprendimento" nella forma "$\Delta W_{ij} = \ldots$", definiamo una *mutazione genetica* intesa come piccolo cambiamento casuale di alcuni pesi W_{ij} della rete o come aggiunta/rimozione di una unità. Definiamo inoltre un criterio di riproduzione e di morte della rete: il sistema mantiene in vita una popolazione di reti che nascono, si riproducono e muoiono. Le reti sono ermafrodite e le figlie sono copie delle madri leggermente modificate per mutazione genetica. Durante la vita ogni rete esegue ripetutamente gli esempi e si riproduce più volte. La longevità e la prolificità sono funzioni dell'errore che la rete commette nell'eseguire gli esempi: chi fa meno errori si riproduce di più. Esiste un limite massimo al numero di reti viventi, raggiunto il quale la popolazione non cresce più e il tasso di natalità diventa uguale al tasso di mortalità.

Facendo evolvere un sistema di questo tipo si avrà inizialmente una popolazione molto varia (reti molto casuali e diverse fra loro) che tende man mano ad un miglioramento della specie grazie alla selezione naturale che privilegia le reti che fanno meno errori. Viene quindi raggiunta asintoticamente la situazione in cui la popolazione è composta da individui tutti perfetti e uguali (a meno delle piccole variazioni genetiche). Un campione di questa popolazione è una rete che ha appreso gli esempi. Si noti il fatto che l'apprendimento è frutto di selezione dalla casualità. I modelli genetici, sebbene affascinanti, non hanno finora avuto molto successo, soprattutto per le eccessive risorse di calcolo e di memoria richieste. Sono attualmente in fase di studio le potenzialità di questo metodo per la realizzazione di reti che automodificano la propria topologia.

9 Applicazioni e strumenti

Le reti neurali sono una tecnologia "orizzontale" che trova applicazioni dall'industria alla finanza, dalla medicina alla difesa, dalle telecomunicazioni alle assicurazioni. Ovviamente non ha senso impiegare reti neurali dove gli attuali computer si sono dimostrati eccellenti (calcoli, basi di dati, word-processing...). Le reti neurali non devono essere intese come un'alternativa al computer, bensì un complemento: è opportuno adottarle dove i computer falliscono, in particolare nei problemi caratterizzati da informazioni incomplete, vaghe, contradditorie, distorte, imprecise, o in quei campi in cui manchi un algoritmo adeguato o infine dove necessiti una velocità tale da richiedere parallelismo. Quindi:

- riconoscimento/classificazione di immagini, testi, voce, segnali, pattern;
- filtri di rumore su segnali;
- sintesi di parlato;
- pianificazione/ottimizzazione;
- controllo di processo e di qualità;
- supporto alle decisioni;
- analisi finanziaria;
- diagnostica medica e industriale;
- previsioni metereologiche;
- compressione/trasmissione di dati;
- gestione database.

Nell'ambito di alcuni progetti di ricerca e industriali si sono sperimentate con successo tecniche neurali per la realizzazione di:

- Un riconoscitore di caratteri manoscritti, in grado di riconoscere dal 90% al 99% di un testo, a seconda della qualità della calligrafia.
- Un riconoscitore di caratteri Kanji, con una accuratezza del 92% su una base di 2000 caratteri.
- Un autenticatore di firme, i cui casi di falso allarme ammontano finora al 4% (prestazione assai migliore rispetto alle tecniche tradizionali).
- Un rivelatore di carte di credito rubate (Chase Manhattan Bank).
- Un classificatore di segnali radar, in grado di raggiungere il 100% delle prestazioni richieste, anche dove le tecniche Bayesiane non superavano il 95%).
- Un rivelatore di esplosivi in servizio dal 1988 presso l'aereoporto di New York, Los Angeles e S. Francisco (tecnica back propagation, controlla 10 valige al minuto, realizzato solo con prodotti commerciali).
- Un sistema di guida autonoma di automobili, in grado di guidare a circa 5 Km/h nei viali alberati della Carnegie Mellon University in diverse condizioni di luce (input=immagine della strada; output=posizione del volante; tecnica back propagation; addestramento effettuato con 1200 immagini in 40 epoche, tempo di addestramento: 30 minuti su supercomputer Warp; tempo di esecuzione: 200 millisecondi su Sun-3; conoscenza sviluppata nelle unità nascoste: "bada al bordo della strada"; vedi figura 26).
- Un sistema di controllo di un robot che governa l'orientamento di un satellite.
- Un lettore di testi ad alta voce, in grado di pronunciare parole (inglesi) mai viste prima con accuratezza del 90% e con lo stesso accento del maestro (back propagation, 309 unità, 18.629 connessioni, implementato in "C" su VAX-780, legge 2 caratteri al secondo).

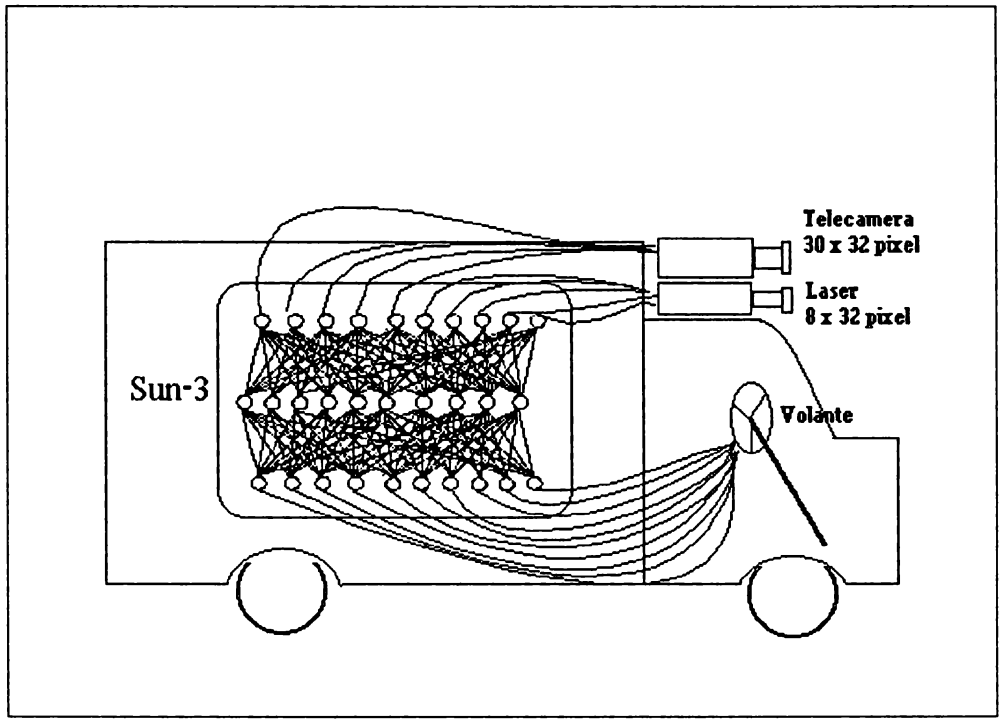

Figura 26. *Applicazione di una rete neurale come sistema di guida autonoma di veicolo.*

- Un riconoscitore di oggetti sottomarini via Sonar (Bendix Aerospace).
- Un correttore di eco in linee telefoniche.
- Un "dattilografo" che trascrive sotto dettatura qualsiasi tipo di testo pronunciato in finlandese o in giapponese con un'accuratezza compresa fra l'80% e il 97% (preprocessing con filtro passa basso, convertitore analogico-digitale, rete auto-organizzante con 15 unità in input e 96 in output, elaborazione in tempo reale su PC/AT con signal processor TMS-32010).
- Un sistema per la concessione di prestiti per l'acquisto di beni di consumo a privati (back propagation, 100 unità di input, 1 unità di output, apprendimento con 270.000 casi di prestiti effettuati nell'arco di 6 mesi, prestazioni migliori del 7% rispetto al sistema pre-esistente di credit scoring).

Strumenti di sviluppo per Reti Neurali

Gli strumenti per la realizzazione di reti neurali si dividono in tre categorie:

- strumenti software;
- schede hardware per computer;
- circuiti ad hoc.

I simulatori software sono strumenti che mettono a disposizione un linguaggio per la definizione di una rete neurale: si può scegliere un modello di rete con architettura prefissata (back propagation, memoria associativa, rete competitiva, macchina di Boltzmann...) oppure si può costruire una architettura ad hoc, definendo il tipo di unità, di connessione, di funzione di trasferimento... Una volta definita l'architettura è possibile istanziare una rete con un certo numero di unità e di livelli e poi farle apprendere un insieme di esempi, eventualmente definendo una formula di apprendimento ad hoc. I prodotti più sofisticati permettono la "traduzione" della rete addestrata in un linguaggio di programmazione tradizionale (generalmente il "C") e l'integrazione della rete in software applicativo; in questo caso la rete viene vista come una sorta di "subroutine" richiamabile all'occorrenza. Sono estremamente utili i programmi di utilità per il debugging e il set-up della rete: grafici dell'errore, mappe di attività, diagrammi dei pesi...

In conclusione, i pacchetti software per la simulazione delle reti neurali (Neuralworks, Neuroshell...) sono ottimi strumenti per la prototipazione e la realizzazione di applicazioni limitate; il loro basso prezzo li può giustificare anche in veste tutoriale. I punti forti sono l'estrema flessibilità, la completezza dei modelli a disposizione, la semplicità di uso, la disponibilità su PC e il basso prezzo. I punti deboli sono il limite di memoria e le prestazioni. Il vero problema della simulazione di reti neurali via software non è la lentezza (una Sun-Sparc ben programmata può superare circuiti hardware artigianali) né la limitazione della memoria RAM, ma è la capacità dei compilatori e dei sistemi operativi di "digerire" tabelle di enormi dimensioni (una rete di Hopfield applicata ad un'immagine di 100x100 pixel richiede una tabella di 100 milioni di elementi).

Una soluzione a questo problema sono le schede hardware aggiuntive per computer: queste generalmente contengono parecchi megabyte di memoria e diversi processori e co-processori matematici. La rete viene comunque eseguita (e addestrata) in maniera sequenziale, infatti queste schede non contengono una circuiteria ad hoc parallela, ma semplicemente dei processori alla Von Neumann con un'architettura ottimizzata per il calcolo matriciale (pipeline). Per questo motivo queste schede prendono il nome di "acceleratori neurali". Esistono in commercio schede di tipo add-on per PC o Sun o Vax (DASH! 860, Anza, NDS, Parallon, Sigma, Mark...) corredate di software per la definizione e istanziazione di reti con architettura predefinita o personalizzata. Le capacità di memoria consentono di lavorare tranquillamente con centinaia di migliaia di unità, milioni di connessioni ad una velocità di esecuzione di decine di migliaia di connessioni al secondo. L'unica forma di parallelismo è il fatto che l'applicazione neurale può girare sulla scheda contemporaneamente ad applicazioni tradizionali operanti sulla CPU del computer ospite; generalmente l'applicazione neurale è vista come un co-processo.

In conclusione, queste schede neurali sono una via di mezzo fra lo strumento prototipale (caratterizzato da alta flessibilità e generalità) e lo strumento applicativo (caratterizzato da alte prestazioni). L'unico punto debole è il prezzo, che spesso supera il valore dell'intero computer ospite. Le schede neurali sono da intendersi come strumenti di studio e sviluppo, non come prodotti per l'esecuzione di applicazioni. Il loro alto costo e la complessità hardware/software le rende improponibili per applicazioni da rilasciare su vasta scala (usando una terminologia software, si potrebbe dire che manca la versione di "delivery" o il "run time").

Un'altra possibilità è l'implementazione di simulatori neurali su macchine intrinsecamente parallele (Transputer, Array Computer, Connection Machine...). Qui vi è il vantaggio di ottenere un reale parallelismo, ma possono sopraggiungere difficoltà tipicamente dovute al fatto che l'architettura di queste macchine è concepita per scopi diversi, che si scontrano con le esigenze neurali.

La strada più promettente, anche se la più difficile, è quella della realizzazione di circuiti ad hoc. Svariati approcci sono stati

tentati: negli anni 60 H.D.Block costruì un perceptrone [27] con un'area sensoriale di 20x20 rivelatori, una rete associativa di 512 unità e 8 unità di output. La rete era fisicamente collocata in una scatola metallica e l'intensità delle connessioni veniva variata, durante l'apprendimento, da potenziometri guidati da piccoli motori elettrici. Questa rudimentale macchina aveva la stessa velocità di una rete neurale simulata oggi su una macchina tipo VAX.

Nel 1984 Hopfield propose una memoria associativa in cui le unità hanno uno stato di attivazione continuo anziché binario [28]: in questo caso, ogni unità è assimilabile ad un amplificatore elettronico di segnali, le connessioni sono normalissimi fili e i pesi sono resistenze elettriche (figura 27). In altre parole, Hopfield propose la realizzazione elettronica di quello che è il neurone biologico: ogni unità è caratterizzata da una capacità di membrana C_i e una resistenza di membrana R_i. L'unità si comporta come un semplice circuito elettrico con una parte lineare RC e una non lineare (la funzione di trasferimento T).

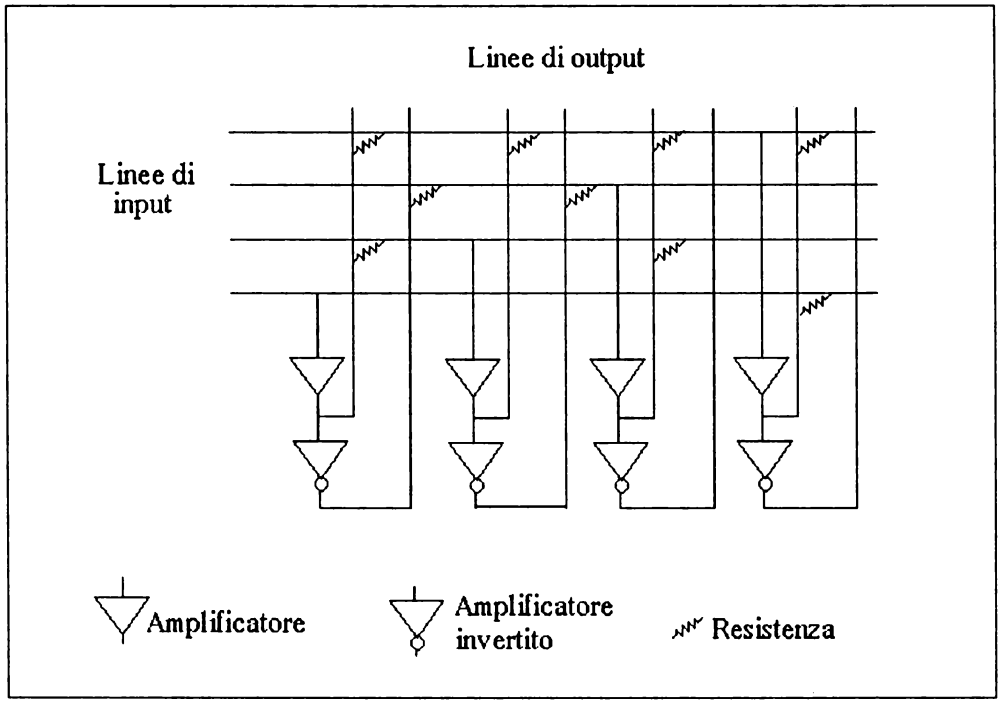

Figura 27. *Un circuito elettrico corrispondente ad una rete neurale di Hopfield con unità ad attivazione continua.*

L'attivazione delle unità è quindi governata da queste relazioni:

$$A_i = \left(\sum_j W_{ij} O_j\right) \cdot \frac{R_i}{1+sR_iC_i}$$

$$W_{ij} = \frac{1}{R_{ij}}$$

Si noti che il peso W_{ij} delle connessioni è l'inverso della resistenza elettrica R_{ij} fra gli amplificatori i e j. L'output delle unità è assimilata alla frequenza di scarica ed è governata da una funzione tipo sigmoide (diversa da quella vista per le reti back propagation):

$$O_i = \frac{1}{2}\left(1+\tanh\frac{A_i}{\lambda}\right)$$

Si noti, in figura 27, che ogni unità è rappresentata da una coppia di amplificatori per poter rappresentare sia eccitazioni che inibizioni fra le unità (poiché non esiste una resistenza elettrica negativa, si applica una resistenza positiva ad un segnale invertito da un apposito amplificatore). Hopfield ha dimostrato che un siffatto circuito manifesta le proprietà tipiche delle reti neurali ed è equivalente ad una rete con stati binari e attivazione stocastica. Svariate implementazioni sono state realizzate e poste in commercio (AT&T, Bell, Synaptics, Fujitsu...). Le realizzazioni su silicio attuali contengono alcune centinaia di unità totalmente connesse, per un totale di centinaia di migliaia di transistor; la tecnologia può essere sia digitale che analogica.

Si stanno diffondendo sul mercato anche sistemi applicativi specifici, corredati di una rete neurale già addestrata o comunque predisposta per addestramenti specifici. Dagli Stati Uniti provengono sistemi per la comprensione di caratteri manoscritti; l'Europa (Università di Helsinki) è particolarmente forte nella realizzazione di dispositivi in grado di scrivere sotto dettatura.

Riassumendo, diverse implementazioni hardware ad hoc sono state realizzate per le reti neurali, tutte caratterizzate da due problemi principali: il primo è che un circuito ad hoc non potrà mai essere general-purpose (ad esempio, un circuito per una memoria associativa non potrà mai funzionare come back propagation). Il secondo è l'oggettiva difficoltà nel creare un "groviglio di fili" che non si tocchino (le connessioni).

Un approccio molto promettente per risolvere quest'ultimo problema è l'impiego di tecnologia ottica anziché elettronica: le proprietà della luce, infatti, calzano a pennello per le funzioni neurali. Come è noto, due fasci di luce che si incrociano non interferiscono fra di loro, ma si "attraversano" senza disturbarsi; inoltre, se diversi fasci di luce convergono in un punto, l'intensità luminosa incidente è la sommatoria delle intensità dei fasci; infine, l'impiego di filtri ottici può fungere da peso sinaptico di connessione. Pertanto, si può ipotizzare la realizzazione di una rete neurale in cui ogni unità sia un dispositivo ottico che riceve alcuni fasci di luce in input opportunamente filtrati, li somma, li elabora tramite lenti e specchi e poi li trasmette ad altri dispositivi senza preoccuparsi del fatto che i loro percorsi si incrocino. Svariati centri di ricerca stanno lavorando su quest'idea, che sembra promettere bene nonostante le difficoltà tecniche (voluminosità delle apparecchiature, dispersione di energia...).

L'ultima frontiera nella realizzazione di reti neurali è l'approccio biologico, in cui ogni unità è effettivamente formata da cellule opportunamente sintetizzate. La prospettiva di poter avere dei "bio-computer" è molto allettante perché potrebbe aprire nuovi orizzonti tecnologici, ma a tutt'oggi è più sogno che realtà.

Bibliografia

[1] J.P. Changeux: "L'homme neuronal", Parigi, 1983

[2] Oxford Companion to Mind: "Memory: biological basis", Oxford, 1987

[3] D.E. Rumelhart, J.L. McClelland: "Parallel Distributed Processing", MIT Press, 1988

[4] K.S. Lashley, "In search of the Engram", Society of Experimental Biology Symposium, Cambridge University Press, 1950

[5] W. McCulloch, W. Pitts: "A logical calculus of the ideas immanent in nervous activity" Bulletin of Mathematical Biophysics, 1943. Raccolto nel volume "Neurocomputing: foundations of research" eds. J. Andersen, E. Rosenfeld, MIT Press 1988.

[6] D.O. Hebb: "The organization of behaviour", New York 1949

[7] F. Rosenblatt: "The Perceptron: a probabilistic model for information storage and organization in the brain", Psycological Review 65, 1958

[8] M.L. Minsky, S.A. Papert: "Perceptrons", MIT Press, 1969

[9] D.E. Rumelhart, G.E. Hinton, R.J. Williams: "Learning internal representations by error propagation" da "Parallel Distributed Processing" di Rumelhart, McClelland, MIT Press, Cambridge MA, 1986

[10] J.P. Hopfield: "Neural Networks and physical systems with emergent collective computational abilities" Proceedings of the National Academy of Sciences 79, 1982.

[11] T. Kohonen, "Associative memory: a system theoretic approach", Springer Verlag, Berlino, 1977.

[12] D. Parisi: "Intervista sulle reti neurali", Ed. Il Mulino, Bologna, 1989

[13] A. Mazzetti: "Reti Neurali: 10 risposte a 10 domande", Informatica Oggi N. 62, Jackson Ed. Giugno 1990

[14] B. Kosko: "Bidirectional Associative Memories", IEEE Transactions on Systems, Man and Cybernetics, 1987

[15] J.J. Hopfield: "Neural Networks and physical systems with emergent collecive computational abilities", Proceedings of the National Academy of Science 79, 1982

[16] R. McEliece, E. Posner, E. Rodemich, S. Venkatesh: "The capacity of the Hopfield Associative Memory", IEEE Trans. on Information Theory, July 1987

[17] R. Hecht-Nielsen: "Theory of Back Propagation Neural Networks" International Joint Conference on Neural Networks, 1989

[18] G.Hinton, T.J.Sejnowski: "Learning and relearning in Boltzmann Machines" in "Parallel Distributed Processing" Rumelhard, McClelland, 1986

[19] S.Kirkpatrick, C.D.Gelatt, M.P.Vecchi: "Optimization by simulated annealing" in "Science", 220, 1983

[20] F.Crick, G.Mitchinson: "The function of dream sleep" in "Nature", 304, 1983

[21] T.Kohonen: "Self-Organizing formation of topologically correct feature map", Biological Cybernetics, 43, 1982

[22] S.Grossberg: "Competitive Learning: from interactive activation to adaptive resonance" in "Cognitive Science", 11, 1987

[23] D.E.Rumelhart, D.Zipser: "Feature discovery by competitive learning" in "Cognitive Science", 9, 1985

[24] C.Von Der Malsburg: "Self organization of orientation senstive cells in the striate cortex", in "Kibernetik", 14, 1973

[25] G.N.Edelman: "The theory of natural group selection" Basic Books, New York, 1987

[26] S.Nolfi, J.L.Elman, D.Parisi: "Genetic adaption and individual learning in neural networks", Institute of cognitive Science, S.Diego University of California, 1989

[27] H.D.Block: "The perceptron: a model for brain functioning", Reviews of Modern Physics, 1962

[28] J.J.Hopfield: "Neurons with graded response have collective computational properties like those of two state neurons", Proc. of National Academy of Science USA, 1984

Finito di stampare nel mese di
Agosto 1990 da Grafica 85 srl
Rodano Millepini

- ☐ *1.000 consigli e trucchi per AutoCAD Cod. 0021 - L. 76.000 - Pag. 512*
- ☐ *600 consigli e trucchi per Commodore Cod. 0031 - L. 48.000 - Pag. 368*
- ☐ *AutoCAD 3D Cod. 0029 - L. 48.000 - Pag. 320*
- ☐ *AutoLISP per AutoCAD 11 Cod. 0049 - L. 38.000 - Pag. 224*
- ☐ *Convertire in OS/2 i programmi in C del DOS Cod. 0020 - L. 53.000 - Pag. 320*
- ☐ *Dati, Relazioni & Associazioni Cod. 0053 - L. 25.000 - Pag. 190*
- ☐ *dBASE IV e dBASE SQL Cod. 0016 - L. 48.000 - Pag. 320*
- ☐ *Dizionario Acronimi & Termini di Informatica Cod. 0012 - L. 48.000 - Pag. 448*
- ☐ *Dizionario Acronimi & Termini di Informatica software Cod. 0013 - L. 90.000*
- ☐ *DOS, Lotus 1-2-3, WordStar, Word, dBASE III Plus Cod. 0024 - 48.000 - Pag. 384*
- ☐ *Il libro dei FAX Cod. 0022 - L. 25.000 - Pag. 212*
- ☐ *HyperTalk & HyperText Cod. 0037 - L. 53.000 - Pag. 464*
- ☐ *Lotus 1-2-3 Cod. 0033 - L. 19.000 - Pag. 160*
- ☐ *Page Maker 4 per Macintosh e Windows Cod. 0056 - L. 53.000 - Pag. 352*
- ☐ *Pal Cod. 0052 - L.58.000 - Pag. 432*
- ☐ *Paradox 3 Cod. 0019 - L. 68.000 - Pag. 512*
- ☐ *Programmare Windows 3 Cod. 0054 - L. 76.000 - Pag. 576*
- ☐ *Reti Neurali Artificiali Cod. 0060 - L. 25.000 - Pag. 128*
- ☐ *Quattro Pro 2 Cod. 0038 - L. 76.000 - Pag. 624*
- ☐ *Sistemi di Supporto alla Decisione Cod. 0018 - L. 48.000 - Pag. 288*
- ☐ *System 7 (e Quick-Time) Cod. 0058 - L. 38.000 - Pag. 320*
- ☐ *Unix per DOS Cod. 0030 - L. 48.000 - Pag. 352*
- ☐ *Ventura 2 - Estensione professionale Cod. 0015 - 68.000 - Pag. 512*
- ☐ *Ventura 3 - Il grande manuale Cod. 0055 - L.68.000 - Pag. 576*
- ☐ *Videotel Cod. 0023 - L. 38.000 - Pag. 272*
- ☐ *Windows 3 Cod. 0028 - L. 25.000 - Pag. 144*
- ☐ *WordStar 5 Cod. 0011 - L. 48.000 - Pag. 352*
- ☐ *Word per Windows Cod.0035 - L. 48.000 - Pag. 352*
- ☐ *Works 2 Progetti Cod. 0060 - L. 33.000 - Pag. 192*

Ogni mese 2 libri novità sui prodotti informatici più interessanti:
richiedete il catalogo con le novità di quest'Autunno
(DOS 5, Borland C++, UNIX 4, HyperCard, AutoCAD...),
ad Apogeo, via Voghera 11/a, 20144 Milano

Inviare in busta chiusa a:
Apogeo, Via Voghera 11/a - 20144 Milano
oppure comunicateci i vostri dati
via telefono, **02-89404722, 02-89408423,**
oppure via fax, **02-89404595.**

Nome _____

Cognome _____

Azienda _____

Professione _____

Indirizzo _____

CAP ____ Città _____ PV ____

☐ Tenetemi aggiornato sulle prossime pubblicazioni Apogeo

Consigli, note, suggerimenti, richieste...

ALESSANDRO MAZZETTI

Una nuova sfida tecnologica

Reti Neurali Artificiali

Introduzione ai principali modelli e simulazione su Personal Computer

Nonostante l'argomento estremamente innovativo, questo testo è orientato al vasto pubblico. L'intento è quello di divulgare e portare a conoscenza di tutti una sfida scientifica e tecnologica che vuole ricostruire con mezzi artificiali le entità cerebrali degli esseri viventi. È un argomento interdisciplinare che supera l'area dell'informatica e sconfina nella biologia, psicologia, fisica, matematica, ingegneria.

Dopo una panoramica introduttiva sulla natura del cervello biologico vengono presentate le reti neurali artificiali e spiegate le loro potenzialità, anche in confronto con gli attuali computer. Nonostante il linguaggio semplice e comprensibile, vengono descritti con rigore e dettaglio i modelli di rete più significativi, citando anche alcuni teoremi (con dimostrazione breve e semplificata); nel testo sono inclusi listati in linguaggio Pascal per la simulazione su Personal Computer dei più interessanti modelli di rete neurale; la versione eseguibile di tali programmi è reperibile attraverso le strutture divulgative della rubrica Telesoftware di RAI-Televideo.

Non mancano, infine, riferimenti bibliografici e citazioni di applicazioni accademiche e industriali. I prerequisiti per la comprensione di questo libro sono una preparazione matematica medio-bassa e una minima conoscenza dei linguaggi di programmazione.

L. 25.000

APOGEO

ISBN 88-7303-002-5

9 788873 030027